Multi-level analysis of Malware using Machine Learning

Subash Poudyal

Multi-level analysis of Malware using Machine Learning

by

Subash Poudyal

A Dissertation
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Major: Computer Science

The University of Memphis
June 2021

DEDICATION

To my parents, wife and sons.

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my advisor, Professor Dr. Dipankar Dasgupta, for his continuous support and mentorship of my Ph.D. study. Your guidance helped me throughout the process of research and writing this dissertation. You also taught me how to think wisely and be a leader and gave me ample opportunities to grow myself with soft skills, research, and leadership skills. I am very thankful for your patience, motivation, and delivery of immense knowledge. I feel very fortunate to have you as my advisor and mentor for my Ph.D. study. I am very grateful to have professor Vasile Rus, professor Kan Yang, and professor Myounggyu Won in my dissertation committee, whose feedback and guidance helped me improve my research work quality. I would also like to thank my lab mates at the Center for Information Assurance (CFIA) for their regular support and interaction.

I am thankful to all the faculty staff who inspired and helped me acquire the necessary knowledge and skills throughout my study. I am also grateful to Ms. Lyndsey Rush, Ms. Rhonda Katherine Smothers, and Ms. Corinne L O'Connor for helping me with academic and administrative-related issues.

To The University of Memphis and all those involved in creating and running this advanced degree program, thank you for identifying the need for an expert in the cybersecurity domain.

Last but not least, I would like to thank my family. Thank you to my parents and my brother and sister for supporting and inspiring me to achieve more. My special thanks go to my loving wife, Laxmi. Thank you, Laxmi, for being with me throughout this journey with many ups and downs. Without your faithful support and inspiration, this journey would not have been possible. It will always be a unique and loving memory with my little ones Aaron and Aayan Poudyal, cheering with me and bringing smiles to my face. Love you, my dear wife and little ones, for your patience throughout this journey.

Abstract

Poudyal, Subash. Ph.D. The University of Memphis. August 2021. Multi-level analysis of Malware using Machine Learning. Major Professor: Dr. Dipankar Dasgupta.

Malware analysis and detection is a critical capability every business and organization needs to defend itself against a growing number of cyber threats. For example, ransomware, an advanced form of malware, makes hostage of user's data and asks ransom, usually in crypto-currencies, to remain anonymous. Significant efforts have been undertaken to combat these attacks, but the threat factors are dynamic, and there lacks intelligent approach to defeat them. Thus, my study is focused on designing a defensive solution against this advanced malware, i.e., ransomware. Many tools and techniques exist that claim to detect and respond to malware. However, such methods rely primarily on static features, rigid signatures, and non-machine learning approaches. Recent tools advertise to have used machine learning techniques but often lack the explainable component, often miss the zero-day malware, and have high false positives. A smart artificial intelligence (AI) technique with deep analysis, worthy feature analysis, and selection could have provided a heightened sense of proper security. This study uses an AI-powered hybrid approach to detect ransomware. Specifically, I proposed a deep inspection approach for multi-level profiling of crypto-ransomware, which captures the distinct features at DLL (Dynamic Link Library), function call, and assembly levels. I showed how the code segments are correlated at these levels for studied samples. My hybrid multi-level analysis approach includes advanced static and dynamic methods and a novel strategy of analyzing behavioral chains with AI techniques. Moreover, association rule mining, natural language processing techniques, and machine learning classifiers are integrated for building ransomware validation and detection model. Experiments with samples from VirusTotal exhibited that multi-level profiling can better detect ransomware samples among other malware families and benign applications with higher accuracy and low false-positive rate. The multi-level feature sequence can be extracted from most of the applications running in the different operating systems; therefore, I believe that my method can detect ransomware and other malware families for devices on multiple platforms.

Table of Contents

List of Figures

List of Tables

1 Introduction

Malware is any size piece of code or program that can cause minor to significant damage to the user, computer, or network. The damage can be financial, reputation, or psychological. Broadly the malware is classified [64, 110] as shown in Figure 1. Malware analytics deals with different categories of malware written by known or unknown adversaries or entities. These entities range from individual hackers to organized groups, which are self or state-operated.

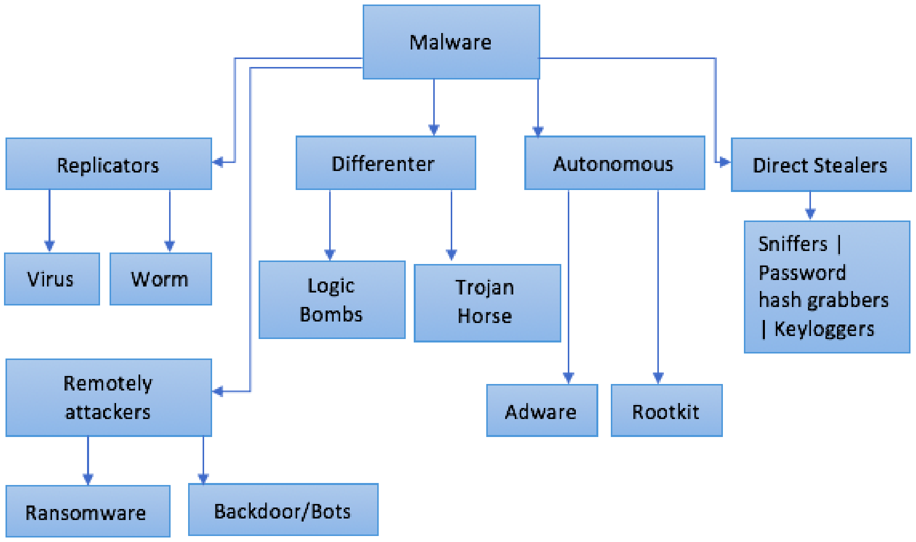


Figure 1: Malware classification [64, 110]

Before I briefly explain each malware categories I want to indicate that these classifications are based on a generalization of essential features; however, each malware category can exhibit the behavior of one or more malware categories.

Viruses and worms fall into the replicators category due to their replication behaviors. The virus attaches to the host program and replicates within the machine or network where it has infected, but the worms can replicate and propagate without the host program in the network. Spammers overload your email box or your web/blog posts with massive amounts of data, which may put your site unresponsive or defaced. In comparison, logic bombs execute when some specific conditions by the malware writer are met. At the same time, Trojan horse may hide in your app (e.g., calculator) and record all your actions (e.g., financial calculations).

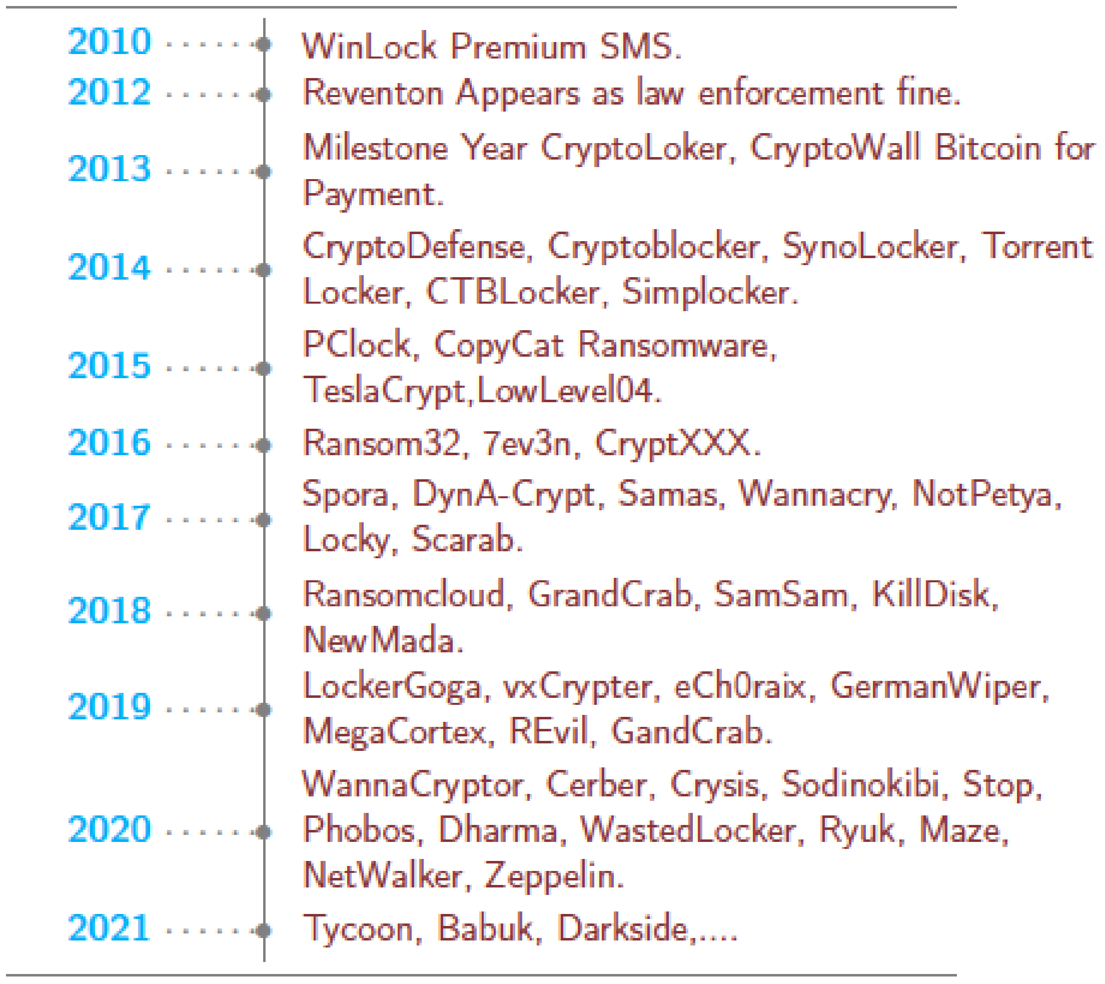
Adware presents unwanted advertisement information and may steal user's information as well. They pop up and try to attract users to click them. Rootkits are malicious code designed to hide the existence of other malware variants. For example, they may hide the presence of the Adwares.

Direct stealers, mainly sniffers, keyloggers, and password hash grabbers try to steal the user's keyword typing events or other system events and capture credentials to online and financial accounts, causing further damage to the victim.

Ransomware is a particular category of malware capable of remote communication with a command and control server for key exchange purposes. They encrypt user's data and hold it until some specified payment is made via cryptocurrency. Backdoors and bots allow the attacker to connect to the victim's computer with little or no authentication and execute commands on the local computer systems.

The first ransomware, AIDS Disk, was first spotted in 1989. The initial version of ransomware used symmetric encryption. Archievus ransomware seen in 2006 was the first to use asymmetric encryption. The most common ransomware families are determined based on different reports from security and cyber defense companies, which include Kaspersky, SonicWall, WeLiveSecurity, and others [1, 2, 34, 37]. Locky, TeslaCrypt, Cerber, GandCrab, Locker, WannaCryptor, TorrentLocker, Locker, WannaCry, Stop, CryptoJoker, Dharma, and CrypoWall are the most common ones among the others. From 2006 to 2015, various versions of both symmetric and asymmetric ransomware appeared. However, table 2 shows the timeline of ransomware for the last 10 years only [33, 39, 41].

Figure 3 shows ransomware families reported in the last three years based on attack frequencies. These ransomware attacks and families are determined based on differ-



2010	WinLock Premium SMS.
2012	Reventon Appears as law enforcement fine.
2013	Milestone Year CryptoLoker, CryptoWall Bitcoin for Payment.
2014	CryptoDefense, Cryptoblocker, SynoLocker, Torrent Locker, CTBLocker, Simplocker.
2015	PClock, CopyCat Ransomware, TeslaCrypt,LowLevel04.
2016	Ransom32, 7ev3n, CryptXXX.
2017	Spora, DynA-Crypt, Samas, Wannacry, NotPetya, Locky, Scarab.
2018	Ransomcloud, GrandCrab, SamSam, KillDisk, NewMada.
2019	LockerGoga, vxCrypter, eCh0raix, GermanWiper, MegaCortex, REvil, GandCrab.
2020	WannaCryptor, Cerber, Crysis, Sodinokibi, Stop, Phobos, Dharma, WastedLocker, Ryuk, Maze, NetWalker, Zeppelin.
2021	Tycoon, Babuk, Darkside,....

Figure 2: Timeline of ransomware families

ent reports from security and cyber defense companies, which include Kaspersky, SonicWall, WeLiveSecurity, and others [1, 2, 34, 37]. Locky, TeslaCrypt, Cerber, GandCrab, Locker, WannaCryptor, TorrentLocker, Locker, WannaCry, Stop, CryptoJoker, Dharma, and CrypoWall are the most common ransomware families among others.

Table 1 shows different recent crypto-ransomware families with properties such as propagation strategy, date when it first appeared, cryptographic techniques used, and command and control (C&C) server used. According to this table, we can infer that most recent ransomware families use the asymmetric encryption method

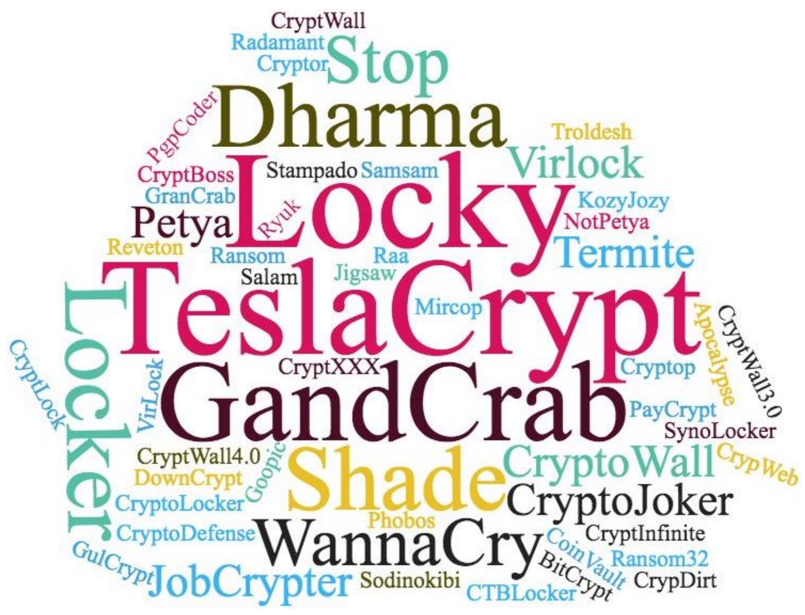


Figure 3: Recent ransomware families(font size based on attack frequency) [1, 2, 34, 37]

to make the decryption task difficult for the victim. The propagation strategy has been consistent with advanced social engineering tactics and exploits tools.

1.1 Terminology

Ransomware/Malware has evolved to become sophisticated malware, and reverse engineering is becoming more challenging. The tug-of-war between defenders and malware writers has created a continuous research scope. Most ransomware attacks occur due to phishing emails which is a type of social engineering attack. This attack can trigger a Denial of Service attack (DoS) or Distributed Denial of Service attack (DDoS) attack and cause data breaches. A recent study shows that machine learning techniques are more effective for malware detection.

Botnets typically cause DDoS attacks. A botnet is a network of devices that an attacker has taken hostage to perform DoS or DDoS attack. DDoS attacks cause a system to become unavailable as it cannot handle the flooding of requests from multiple sources or devices. Due to this, the target machine cannot accept the le-

Table 1: Recent ransomware families with some properties

Family	Propagation strategy	Date appeared	Cryptographic technique	C&C Server
Cerber	Email spam, RIG and magnitude exploit kit	2015	RC4 and 2048-RSA	IP range
TeslaCrypt	Angler browser exploit kit	2015	AES-256	Tor anonymity network
Shade	Malicious websites, exploit kits, infected email attachments	2015	256-AES	Tor anonymity network
Locky	Spam campaigns, Neutrino exploit kit, Nuclear exploit kit, RIG exploit kit	2016	RSA and AES	Using DGA algorithm
Dharma	Unprotected RDP port, Spam campaigns	2016	256-AES and 1024-RSA	Random IPs/locations
Stop	Malicious email attachments/advertisements, torrent websites	2018	256-AES and 1024-RSA	Listed address
GandCrab	Spam emails, exploit kits	2018	AES and RSA	Tor anonymity network/DGA
Ryuk	TrickBot and RDP	2018	AES and RSA	Internet-facing Mikrotik router
Anatova	Spear phishing in private p2p network	2019	RSA and Salsa20	Listed address
Maze	Malspam campaign, RDP attacks	2019	ChaCha20 and RSA	Tor network
AgeLocker	Age encryption tool of Google	2020	X25519 (an ECDH curve), ChaChar20-Poly1305, HMAC-SHA256	Tor network
Snake	Malspam campaign, RDP attacks	2020	AES-256 and RSA-2048	Listed address
WastedLocker	Fake browser update	2020	AES-256 and RSA-4096	Tor network
Conti	Phishing emails, Server Message Block	2020	AES-256 and hard-coded public key	Listed address range
Babuk	Spear phishing	2021	ChaCha and Elliptic Curves	NA
Darkside	Spear phishing, unpatched vulnerability	2021	Salsa20 and RSA-1024	Listed address

gitimate traffic and the service goes down. There was a significant increase in the number of DDoS attacks during the COVID-19 period [3]. The attack trend can be observed going upward in the graph shown in Figure 4.

Phishing is a social engineering attack technique where a user is lured to click a malicious link in the email, document, blog post, or relay channels. The malicious link is specially crafted in a seemingly legitimate email or posts that the user may be very genuine but gets trapped. This user action allows the malware to be downloaded into your system and start its malicious behavior.

Reverse Engineering [74] is taking a binary file that is meant to be read by the computer and using the opcodes to generate assembly, and then reading that assembly to help accomplish whatever goals we may have. I performed the reverse engineering using static analysis of the ransomware and normal binaries leveraging the existing disassembler objdump [112], PE parser [31] and other advanced techniques.

Machine learning allows a system to learn from data rather than through explicit programs. Various supervised and unsupervised algorithms have been used to train

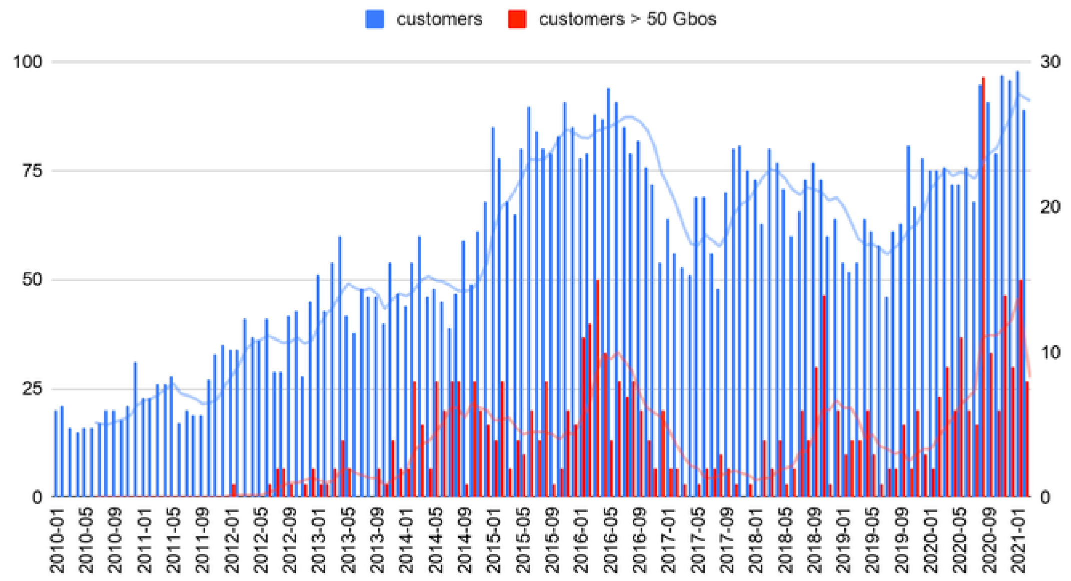


Figure 4: DDoS attack on the increasing trend [3]

the data, which helps to create a predictive model. This model then predicts the outcome for the new or unseen sample. Machine learning approaches are extensively used in various domains ranging from cybersecurity, health, education, business, and space explorations.

Advanced Persistent Threat (APT) is a sophisticated technique used by an attacker to remain undetected for a prolonged period causing significant damage to the victim's resources. More organized criminals and gangs use APT to get into the system and do lateral movement to find critical servers to launch further attacks. The goal may be to steal data or lock the data/system for financial gains.

1.2 Malware Components

Malware comes in various forms with dynamic and unpredictable behavior. Most malware families have the following components, which make malware fully functional.

1.2.1 Payload

The payload is the code portion in a malware executable responsible for malicious action such as deleting, blocking, or encrypting data, sending spam, and so on. The remaining is the overhead code which does tasks such as spreading over the network and avoiding detection.

1.2.2 Packer

A packer compresses the data for various reasons, such as easy transfer over the network and avoids detection by anti-virus or monitoring tools. It uses several data compression algorithms, which are: APLib, LZMA, LZSS, and ZLib. Malware writers use packers to obfuscate their code to thwart detection during infection, installation, execution, and propagation. The bad guy can encrypt the code and then pack it to make the detection more difficult. In this case, even if the defender unpacks the code, they still need to decrypt the code. A decompression stub is generally added at the entry point of the executable. This stub is a piece of code to decompress the compressed data.

1.2.3 Persistence

Every malware tries to be persistent through every boot of the system to fulfill its bad intention. Windows allows a program to start when it boots if:

- The program is kept inside the startup folder.
- The absolute path of the program points to the key in the registry database.
- Some program run as a service under the svchost.exe process
- Programs absolute path is included in the certain batch and init files such as autoexe.bat, wininit.ini, and winstart.bat.

1.2.4 Stealth

Malware tries to hide from victim and anti-virus tools by using various approaches: - Simplest one is hiding the exe extension - Injecting its code into an already running legitimate process (thread injection, Dll injection, process hollowing). The steps in thread injection are shown in Figure 5. Process hollowing is an old technique, but it is getting more popular being adapted by ransomware writers. In process hollowing, a process is run in a suspended mode, and the malicious process is attached to it to resume later. A rootkit is a technique adopted by malware to hide. It does so by modifying a system function or a data structure. Ransomware may use a rootkit to hide the malware that downloads another malware. The next

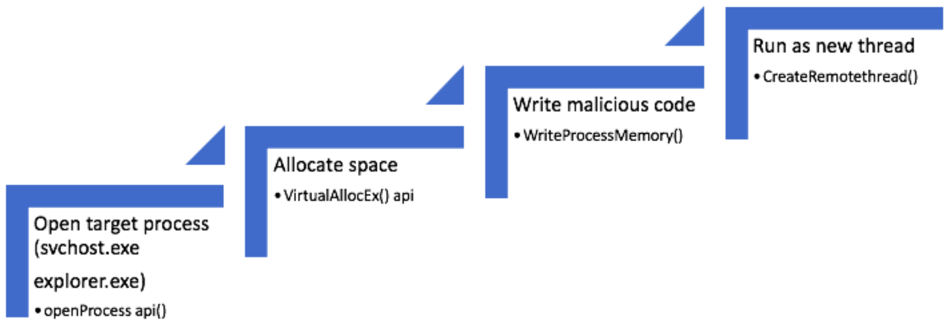


Figure 5: Steps in thread injection

technique to hide is by using a Powershell script where an adversary can download and inject malicious code into a legitimate process's memory. The downloaded malware is never written as a file to the hard disk. For this reason, it is called fileless malware. SoreBrect is ransomware that makes use of fileless property.

1.2.5 Self-defense

Malware defenders utilize various tools to defend against malware attacks. Windows defender, troubleshooting tools, debuggers, system monitoring tools, etc., help track malicious activity. The attackers are notoriously clever and trick the defenders in various ways. For example, they use Microsoft's IsDebuggerPresent() API to detect the debugger; look for files and processes related to ollydbg.exe, idapro.exe, tcpdump.exe, wireshark.exe; look for VMware associated processes, files, and keys in the Windows guest OS on VMware. Sometimes malware is seen using sleep() API to fool VMware by not executing its malicious part. The VMware sandbox environment is generally designed to run for a specific time frame and be restored to a clean state.

1.2.6 Command and control server (C&C Server)

The command and control server (C&C Server) is the control center for the malware, and it is used to send and receive instructions/data between it and the victim machine. Malware receives configuration information and cryptographic keys from the C&C server while sending the stolen data to the C&C server. Previously, IPs and domain names of the C&C server were static and a part of the malware code. These were quickly blocked by the defenders using firewalls and intrusion detection tools. But, recently, malware writers have adopted a Domain Generation Algorithm (DGA), which can generate thousands of domain names that they register for a short period. This made the security analyst and defenders work hard. Figure 6 shows a botmaster controlling different C&C servers, which in turn keep tracks of the infected machines or bots [16]. Broadly, the C&C server can exist in three forms as described below [9].

- **Centralized C&C** : It is a single central server with high bandwidth and processing power under the attackers' control. This server is used to send and receive communications from victimized machines. This model is simple and easy to implement with low latency. The type of communication messages depends upon the nature of malware. The downside of this type is that it can be a single point of failure. A defender once identifies the address of the C&C

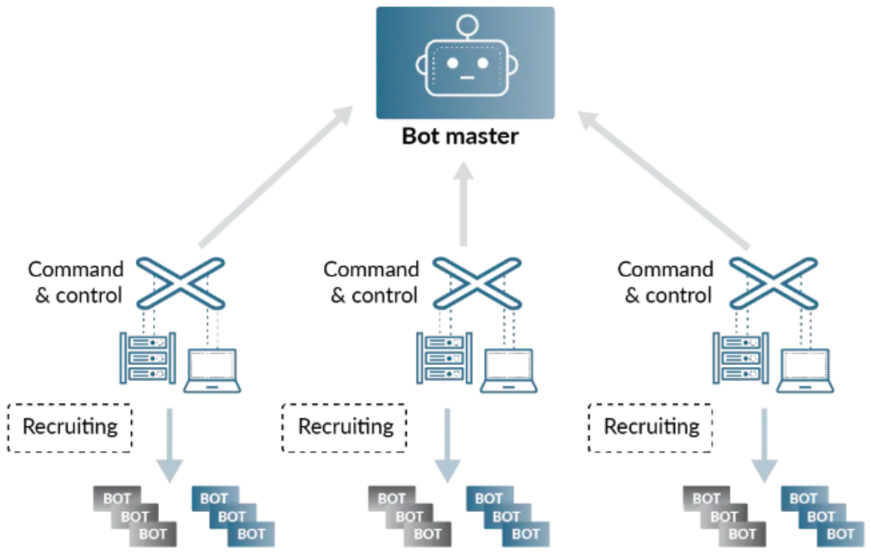


Figure 6: Interaction in C&C Server [16]

server, can block it and thus, prevent the further damages caused by an attack. Example AgoBot and Zotob used this type of server.

- **Peer-to-Peer (P2P) C&C**: This type makes use of the P2P communication protocol. More than one compromised machine takes part in communication with one another. This avoids the central point of detection and failure, making the detection harder for the defender. For example, Phatbot has used P2P communication to control botnets.

- **Random C&C**: Instead of initial establishment of the communication channel, this type has one bot master which scans the internet to find other bots or compromised machine. Once found, the instruction commands are transferred to the victim machine. This type is unpredictable so, the detection is challenging. But, it has a scalability issue. This type is predicted to see in the recent or coming version of the malware.

1.3 Ransomware attack anatomy

The basic anatomy of a ransomware attack consists of five steps as shown in Figure 7

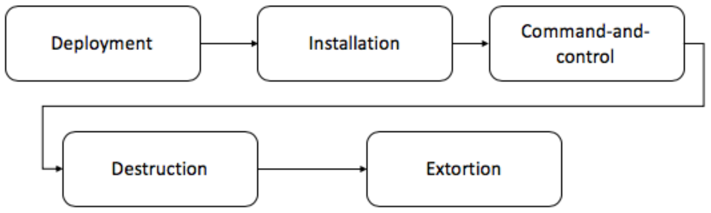


Figure 7: Basic anatomy of Ransomware attack

1.3.1 Deployment

This stage is sometimes also referred to as the infection stage. The first task of a ransomware attack is to get into the victim's system to infect, encrypt or lock the user's data. This is done by various means such as drive-by download, Watering-hole attacks, phishing emails, and exploiting vulnerabilities in internet-connected systems.

- In a drive-by download attack, a malicious code is downloaded automatically and executed without the user's knowledge. It takes advantage of a device or software with security flaws as they were not updated. In October 2017, CryptoLocker ransomware infected Issaquah city of Washington, the USA, where a drive-by download attack was used [22].

- Watering-hole attacks are targeted attacks against individuals or organizations and use drive-by download techniques. There have been several target attacks in US government and private organizations [4, 35].

- Phishing emails are specially crafted emails containing malicious attachments or links. Users are often lured to click or download the extension by tricking a fake application or information to be a real one. For example, CryptoMix that appeared in 2014, uses this technique. Figure 8 shows the workflow of a phishing email that tries to steal the victim's credential to escalate malware attacks further. According to a 2020 digital defense report by Microsoft, among 6 trillion scanned email messages, they were able to block 13 billion malicious emails and 1.6 billion URL-based phishing emails [28].

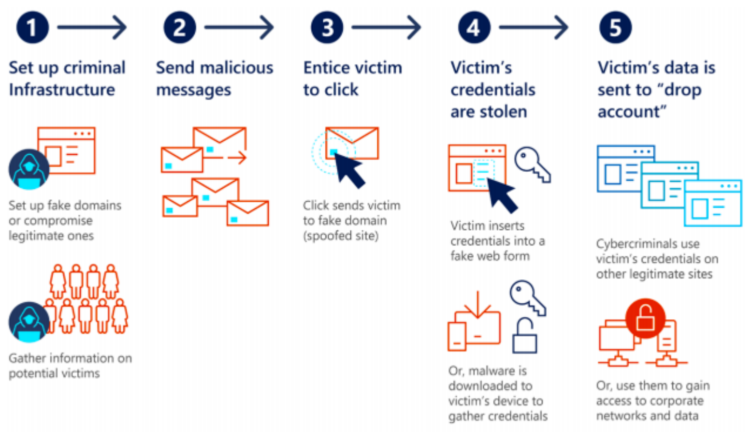

1
Set up criminal Infrastructure
Set up fake domains or compromise legitimate ones
Gather information on potential victims
2
Send malicious messages
3
Entice victim to click
Click sends victim to fake domain (spoofed site)
4
Victim's credentials are stolen
Victim inserts credentials into a fake web form
Or, malware is downloaded to victim's device to gather credentials
5
Victim's data is sent to "drop account"
Cybercriminals use victim's credentials on other legitimate sites
Or, use them to gain access to corporate networks and data

Figure 8: An example of phishing email [28]

- Exploiting vulnerabilities deals with scanning networks for vulnerable devices or machines. These vulnerabilities may be system design flaws, buffer overflow, open ports, miss configurations, and so on. Once identified, the attack is launched. For example, WannaCry uses Windows SMB vulnerability.

1.3.2 Installation

The installation process starts after the delivery of the malware payload.
The payload is usually a download dropper, a piece of code to evade easy detection. Figure 9 shows the ransomware attack pattern from initial setup to delivery of payload [28]. Once this payload is executed, ransomware is downloaded from the C&C server and installed into the victim's system. Ransomware writers try to make the detection difficult by breaking the malware components into differ-

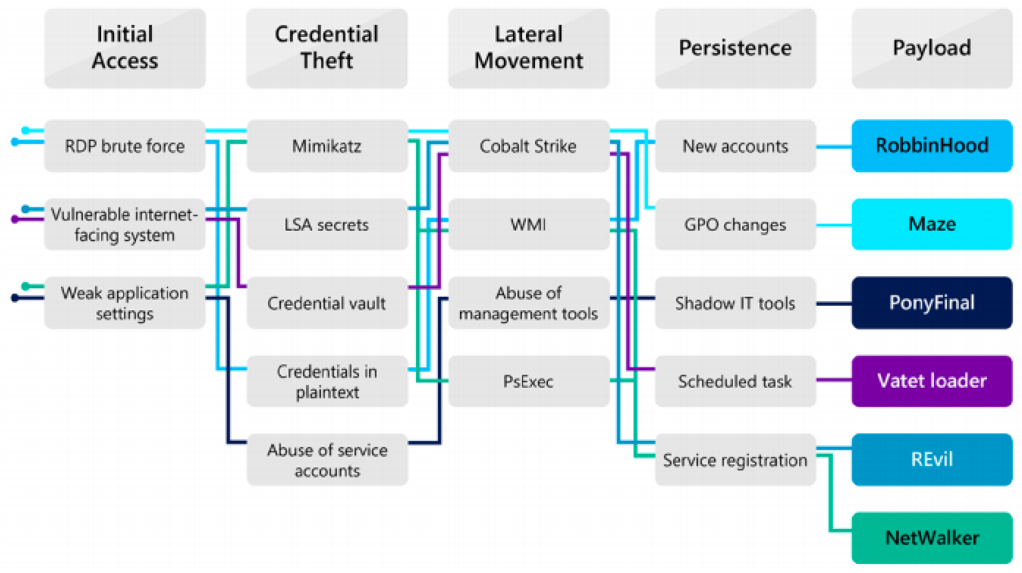


Figure 9: Ransomware attack pattern steps from initial access to dropping payload [28]

ent pieces of scripts, processes, and batch files and may use encrypted and packed codes. It may decide not to execute if it detects processes (e.g., VBoxService.exe, vmtoolsd.exe) and Dlls (e.g., sbieDll.dll) related to Virtualbox machines. Its writers use the MD5 hash of the computer name or a Mac address to identify the victim's machine. It then turns off shadow copy features on files and volumes, turns off system recovery features, and kills anti-malware and logging tools. It then adopts the rootkit technique and attaches itself to the windows process like svchost.exe.

1.3.3 C&C

Some details of the C&C server are discussed previously. Here, we deal more specifically with ransomware. Most ransomware prefers TOR (open-source software to enable anonymous communication) service rather than simple web-based communication. TOR will make a security analyst or defender's job hard as tracing the attack source becomes extremely difficult. TOR clients are often installed on endpoints to ensure secure communications. Once a system gets infected, a prearranged handshake protocol is established among the victim (client) and the C&C (server). The adversary examines the information received from the client to ensure the intended target is who they wanted. This is done via handshake protocol. Ransomware is found to use the same key symmetric encryption to complex asymmetric encryption such as the RSA 4096-bit encryption algorithm. The private key is kept on the server while the selected files are encrypted using the public key on the client-side. Some commonly used C&C servers are Tor/Onion network, centralized server(static IP address or domain name), P2P network, and dynamic server(that generally uses domain generation algorithm).

1.3.4 Destruction

The objective of this phase is to encrypt the victim's files or to lock the system. The attackers define the file types to be encrypted or locked. The malcode starts to encrypt those files. Most of the time, file contents are encrypted, but filenames are also found to be encrypted. This makes the decryption and identification process more difficult. The files that are needed to be destroyed are identified by most usage patterns seeing logs and recent files. Generally, files with .doc, .pdf, .jpeg, .jpg, .png, .xls, .ppt and so on are encrypted and their original files destroyed thereafter.

1.3.5 Extortion

Ransomware attack launchers try to be anonymous by using the TOR network and demand money through crypto-currency in the form of bitcoins. The typical cost to unlock the file is between $300 and $500, but the demand is higher depending upon the target. Some adversaries try to convince by showing one decrypted file, while others not. It is not guaranteed that you will get your original files back even after paying the ransom. The most common extortion techniques are bitcoin and payment vouchers.

1.4 Cryptographic operations

Crypto-ransomware makes use of either symmetric or asymmetric encryption to encrypt the victim's files. Below I discuss why each one is preferred. The recent version of crypto-ransomware uses either asymmetric or a combination of both encryption types.

1.4.1 Symmetric key encryption

This type of encryption uses the same key to encrypt and decrypt the files. Malware writers generate the symmetric key in the victim's machine. This has few advantages. The first is minimum resource utilization and performance overhead. There is a reduced chance of detection as it does not make frequent calls with the C&C server. The generated key is removed and send to the C&C after the encryption is over. The key is given back after the ransom is paid. The decryption can be done either online or offline. The downside of this approach is that the defenders can get the encryption key from memory by analyzing the memory using tools like volatility. For example, Reveton ransomware is based on DES and RSA.

1.4.2 Asymmetric key encryption

This type of encryption method uses both public and private keys, commonly referred to as public-key cryptography. The keys are generated in the C&C server. The public key is passed either by attaching to the payload or send afterward at a suitable time. These approaches make it challenging to get back the files using memory forensics, as seen previously in the case of symmetric key encryption. Embedded public keys require a new public key for each attack, whereas the attacker can use different key pairs for each infection for the downloaded public key. Attackers often use larger primes in their encryption algorithm, for example, RSA 2048bit to 4096-bit.

Recent variants of crypto-ransomware use both symmetric and asymmetric encryption. CryptoDefense ransomware uses AES encryption to encrypt the files. This locally stored symmetric key is then encrypted using a downloaded RSA-2048 public key. After the ransom is paid, the victim is given the private key to decrypt the locally stored symmetric key, which later on can be used to decrypt the files. For example, CryptoWall version 3 ransomware is based on RSA public/private key cryptography and AES in CBC mode.

1.5 Machine Learning

Machine learning is a systematic approach that allows systems to learn and improve from experience without being explicitly programmed automatically. Malware often poses dynamic behavior, due to which signature-based detection is not effective. To better analyze the distinct pattern from the available dataset, I have used machine learning so that the prediction model can help to detect known or unknown samples with acceptable accuracy rates and low false positives.

Machine learning usage is becoming a more widespread technique for malware detection due to the high accuracy rate. Here, I have applied various supervised machine learning algorithms. In particular, I use Bayesian Network, Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, and Adaboosting with different classifiers. A brief discussion of the machine learning classifiers used follows below.

1.5.1 Bayesian Network (BN)

A Bayesian Network represents the joint probability distribution by specifying a set of conditional independence assumptions (represented by a directed acyclic graph), together with sets of local conditional probabilities [95]. Experiments were conducted using a SimpleEstimator [8] to estimate the conditional probability distributions of a Bayes network once the structure has been learned. Alpha is set to 0.5, which is used to estimate the probability tables. I use K2 [69] as a search algorithm. It is a hill-climbing search algorithm that adds arcs with a fixed ordering of variables.

Bayes theorem is the backbone of Bayesian learning methods as it allows to calculate of the posterior probability P(hlD) from the prior probability P(h), together with P(D) and P(D(h). Bayes theorem:

$$P(h \mid D) = \frac{P(D \mid h)\, P(h)}{P(D)}$$

1.5.2 Logistic Regression (LR)

Logistic Regression is the powerful machine learning algorithm used for linear and binary classification problems. I use the multinomial logistic regression model with a ridge estimator [13]. The algorithm is modified to handle the instance weights, and nominal attributes are transformed into numeric attributes using a NominalToBinaryFilter class. This algorithm is optimized by conditional likelihood.

In order to keep the outcome between 0 and 1, the logistic function (sigmoid function) is applied as:

$$g(z) = \frac{1}{1 + exp(-z)}$$

The logistic regression hypothesis is defined as:

$$h/_{\beta}(x) = g(\beta^T x)$$

1.5.3 SMO with Linear Kernel (SMO with LK)

Sequential Minimal Optimization (SMO), invented by John Platt, is an optimization algorithm for training a Support Vector Classifier or Support Vector Machine (SVM). Its implementation globally replaces all missing values and transforms nominal attributes into binary ones. It normalizes all attributes by default where the coefficients in the output are based on the normalized data and not on the original data [14]. SMO is widely used for training Support Vector Machines and is implemented by the popular LIBSVM tool [66]. Experiments were performed using SMO with Linear Kernel and Logistic Regression as the calibrator.

Implementation: We first need to formulate the optimization problem and then compute the support vectors by solving the optimization problem. Recover the weight vector w and the bias b from the support vectors.

$$w = \sum_{j=1}^{k} \alpha y_j x_j$$

$$b = \frac{1}{y_s} - w^T x_s$$

Here, (x_s, y_s) is a support vector.

For classification of point z, compute the sign of $w^T z + b$. If the positive sign, then the class is positive else class is negative.

1.5.4 SMO with Poly kernel (SMO with PK)

This machine-learning algorithm is similar to the one described above. The only difference is that here I use the polynomial kernel of degree two in our experiment.

1.5.5 J48

J48 is the C4.5 [103] algorithm for building pruned or unpruned decision trees. It is the predictive classifier that decides the class of new unseen samples. I choose the pruned decision tree with a confidence factor of 25%. The lower the confidence factor, the heavier the pruning. Also, the minimum number of instances per tree is set to two.

The decision tree uses entropy. Entropy is used to measure the uncertainty in any random variable.

Let $S = (s_1, ..., s_n)$be a partition of the instances based on a feature which can take n values. $p_i = P(s_i)$ is the probability value that an instance has feature value i.

$Entropy(S) = \sum_i p_i log \frac{1}{p_i}$

If any of the subsets s_i is subdivided, the new partition T has a larger entropy than S.

1.5.6 Random Forest (RF)

Random Forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest [62]. One selected subset of training data is used to train each tree with replacement. The remaining subset of training data is used to estimate the error. The generalization error is dependent on the strength of each tree in the forest and the correlation between them. I perform 100 numbers of iterations with one as the minimum number of instances per leaf.

Mathematical description: A random forest is a classifier based on a family of classifiers $h(x|\theta_1), ..., h(x|\theta_k)$ based on a classification tree with parameters θ_k randomly chosen from a model random vector θ.

Given data $D = (x_i, y_i)_{i=1}^n$ I train a family of classifiers $h_k(x)$.

Each classifier $h_k(x) \equiv h(x|\theta_k)$ is a predictor of n.

so $y = + - 1$ which is an outcome associated with input x.

1.5.7 AdaboostM1 with J48 (Ada with J48)

AdaboostM1 [75] is used to tackle the nominal class problem. It often dramatically improves performance but sometimes overfits [12]. AdaboostM1 is used to improve the performance of the learning algorithm. For weak classifiers such as Decision Stump, AdaboostM1 improves the performance significantly. Experiments use the J48 classifier with AdaboostM1.

Some mathematical derivation: I set initial probabilities of training examples as $p_1, p_2, ..., p_N$. Sample a subset S_t of training examples where p_i is the probability of choosing the i-th example. Then apply the weak classifier to S_t to compute hypothesis h_t.

Probability update in t-th iteration $\alpha_t = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t}$ where ϵ_t is the weighted training error.

$q_i = e^{-\alpha_t}$ if $h_t(x_i) = y_i$

$q_i = e^{\alpha_t}$ if $h_t(x_i) \neq y_i$

New $p_i = \frac{p_i q_i}{z_t}$

Then the final classfier becomes: $f_t(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$

1.5.8 AdaboostM1 with Random Forest (Ada with RF)

This algorithm uses the AdaboostM1 discussed above, but along with a Random Forest classifier.

1.5.9 Deep learning

Deep learning is an advanced type of neural network with many layers and performs complex tasks like face recognition, language translation, etc. Deep learning methods have the capability to combine the original features to form new optimized meta-features automatically. These meta-features are again combined to create even more features until the model is robust and achieves the desired accuracy threshold. More details about deep learning is discussed in one of the recent work [61].

1.6 Association Rule Mining

Association rule mining is a data mining technique that finds patterns and relationships among items in a given data. It is referred to as market basket analysis when used with sales data or point-of-sales system in the supermarket. Today, association rule mining is used in many applications, including web mining, intrusion detection, network analysis, etc. In association rule mining, we need to find frequent itemset first. A data item that frequently occurs in a given dataset becomes a part of a frequent itemset. The support count of an item is the frequency of the item in the dataset. Generally, a minimum support value is defined, and those items whose support is less than the minimum support are not considered for further calculation steps. Association rule is defined as an implication of the form of if(antecedent) and then(consequent), which is written as X $\longrightarrow$ Y.

Apriori algorithm is the most basic rule mining algorithm that works based on prior knowledge of frequent itemsets [52]. A minimum support and confidence threshold is set for the algorithm before generating candidate itemsets. Finally, I mine the association rules. However, this approach is computationally inefficient as it requires multiple scans of the database to generate itemset. In contrast, the FP-Growth algorithm, an array-based, uses depth-first search and requires only two database scans, making it more efficient and scalable.

1.7 Natural Language Processing (NLP)

NLP language models have proved helpful in recommendation systems, text classification, speech recognition, and so on. In this work, I have exploited some popular concepts but applied them to a unique problem domain of the multi-level analysis model of ransomware detection. NLP schemes used are composed of three methods: N-gram generation, N-gram probability, and TF-IDF described below.

1.7.1 N-gram Generator

An n-gram model is a type of probabilistic language model that predicts the next possible item in a sequence. N-gram is a contiguous sequence of n items from a given sample of text corpus or speech corpus. In my experiment, it is the DLL, function call, and assembly instruction corpus. The items can be phonemes, letters, words, DLLs, functions call, opcodes, or base pairs depending upon the type of application considered. The value of N can be 1,2,3,4, or any other positive integer. The value for N depends upon the problem domain, type, and nature of the dataset. In text processing tasks, a smaller N usually decreases the classification performance heavily, while the larger N is not considered too relevant. In the experiment, I choose different values of N ranging from 2 to 6 to analyze the detection accuracy of ransomware.
The n-gram generation component is responsible for generating possible sequences of n-grams for a given value of N. I consider only the unique set of n-gram sequences.

1.7.2 N-gram Probability scoring

N-gram Probability scoring component is fed with the n-gram sequences obtained from the n-gram generation component. I apply the Markov assumption by considering only the immediate N-1 words.

In a n-gram, we consider the length $n-1$

$$p(w_i|w_1,\ldots,w_{i-1}) = p(w_i|w_{i-n+1},\ldots,w_{i-1})$$

- unigram: $p(w_i)$
- bigram: $p(w_i|w_{i-1})$ (Markov process)
- trigram: $p(w_i|w_{i-2},w_{i-1})$

We can estimate n-gram probabilities by counting relative frequency on a training corpus.

$$\hat{p}(w_b|w_a) = \frac{c(w_a,w_b)}{c(w_a)}$$

N is the total number of words in the training set and $c(\cdot)$ denotes count of the word or word sequence in the training data.
For the experiment dataset, the n-gram sequence will be the DLL, function call, and assembly instruction sequences. N will be their corresponding total number of sequences in the corpus. The probability scores for each n-gram sequence are stored in a feature database.

1.7.3 TF-IDF

TF-IDF is the product of Term frequency(TF) and Inverse document frequency(IDF). Term frequency is simply the number of occurrences of particular n-gram sequences in a binary sample, whereas the IDF is given as:

$$IDF(ngram) = log_e \frac{Total\ no\ of\ binaries}{No\ of\ binaries\ with\ ngram\ in\ it}$$

1.8 Research contribution

This Ph.D. work resulted in notable contributions to the research community. Following is the list of publications that resulted from this Ph.D. study.

- Dipankar D, Poudyal, S. AI-Powered Advanced Malware Detection System. Patent under submission.
- Poudyal, S, Dasgupta D. Analysis of Crypto-Ransomware using ML-based Multi-level Profiling. Under submission.
- Poudyal S, Dasgupta D. AI-Powered Ransomware Detection Framework. In 2020 IEEE Symposium Series on Computational Intelligence (SSCI) 2020 Dec 1 (pp. 1154-1161). IEEE.
- Poudyal S, Dasgupta D, Akhtar Z, Gupta K. A multi-level ransomware detection framework using natural language processing and machine learning. In 14th International Conference on Malicious and Unwanted Software" MALCON 2019 Oct.
- Poudyal S, Akhtar Z, Dasgupta D, Gupta KD. Malware analytics: review of data mining, machine learning and big data perspectives. In 2019 IEEE Symposium Series on Computational Intelligence (SSCI) 2019 Dec 6 (pp. 649-656). IEEE.
- Poudyal S, Gupta KD, Sen S. PEFile analysis: a static approach to ransomware analysis. Int J Forens Comput Sci. 2019;1:34-9.
- Poudyal S, Subedi KP, Dasgupta D. A framework for analyzing ransomware using machine learning. In 2018 IEEE Symposium Series on Computational Intelligence (SSCI) 2018 Nov 18 (pp. 1692-1699). IEEE.

- Basnet, M., Poudyal, S., Ali, M., & Dasgupta, D. (2021). Ransomware Detection Using Deep Learning in the SCADA System of Electric Vehicle Charging Station. arXiv preprint arXiv:2104.07409, Accepted for: 2021 IEEE Smart-GridTechnologies, Sept 2021.

2 Background and Related Work

The economic benefits and anonymity has fostered cyber-criminals to perform continuous ransomware attacks in various sectors. Recent years have seen attacks being spread as a ransomware-as-a-service model. These attacks are often delivered via phishing campaigns where a user is masqueraded with a seemingly genuine email with a malicious link or attachment. The victim often becomes prey to social engineering attacks and lures to click the link or download the malicious attachment. According to the survey done by Sophos [42], 45% of ransomware attacks are via malicious links or attachments in emails. Also, 21% of the attacks are from a remote attack on the server and the remaining through misconfigured systems, USB devices, etc. Recently, phishing attacks are coming in the form of COVID-19 themed lures and exploit people's concerns over the pandemic and safety of their family members. The bad guys try to increase the anxiety level of internet users. Some of the lures include information about vaccines, masks, and hand sanitizer; insurance plans to cover COVID-19 illness, government assistance forms for economic relief, and critical updates to consumer applications [23]. Other forms of attacks include exploiting vulnerabilities in user systems or in the application they are using. Unpatched systems provide room for remote code execution and privilege escalation. The MS17-010 [29], SMB vulnerability, and EternalBlue exploit made several WannCry ransomware attacks all over the world in 2017 [49]. However, the same vulnerability and exploit is even used today by recent ransomware families, including Ryuk, SamSam, and Satan [40]. This indicates systems that are not patched are prey to continuous attacks disrupting regular businesses/services with financial losses and reputation.

Due to the recent lockdown all over the world and restricted social behavior, people are working remotely. People are using the internet and are busy with social networking and online e-commerce applications more than ever. This increased internet activities and poor network infrastructure, IoT devices, and untrusted web applications are causing people more vulnerable to malware attacks.

On July 23, 2020, Garmin, a multi-national technology company, suffered from a WastedLocker ransomware attack disrupting various customer services, apps, and possible data breach [20]. The customers were unable to log in, record, or analyze their health and fitness data. WastedLocker first appeared in the wild in April and often leveraged the payload through SocGholish and Cobalt Strike tools [47]. The user files are encrypted via AES symmetric keys, which are then encrypted using an RSA-4096 public key.

On May 7, 2021, Colonial Pipeline said that a ransomware attack forced the company to proactively close down operations and freeze IT systems after becoming the victim of a cyberattack [15]. More than 100GB of data got stolen in just 2 hours. Salsa20 and RSA-1024 encryption were used to encrypt user files. Another recent attack includes unauthorized access on the Washington D.C. Metropolitan Police Department (MPD) server where the Babuk Locker ransomware gang claimed the responsibility of the attack after releasing the screenshots of the files and folders [18]. Cybercriminals have demanded an enormous ransom in return for hostage files. The city of Tulsa, Oklahoma, suffered a ransomware attack on the first week of May, 2021 [11]. This forced the city to shut down all of its systems, and most of its online services have been disrupted.

Ransomware often comes in two forms: Crypto ransomware and locker ransomware. Crypto ransomware is more prevalent these days, encrypting a user's data and holding it until the ransom is paid. In contrast, Locker ransomware locks the user's system making the system unusable. The Locker ransomware often replaces the whole screen with a warning picture with instructions to pay to get the system to a normal stage. Hybrid encryption is more common, which includes the combination of symmetric and asymmetric encryption. The user files are encrypted using the locally generated keys, which are encrypted by the attacker's public key. To get the data back, the victim needs the attacker's private key. It may also infect the master boot record and replace the start-up screen with a warning message. In the case of Crypto ransomware, to make the attack vector more sophisticated so that the defenders would not decrypt the user's data by breaking into the algorithm used, ransomware writers adopt advanced encryption techniques.

Ransomware attack vectors are dynamic and use sophisticated encryption techniques. They come in various obfuscated and persistent forms making the analysis and detection work more complex. Various anti-ransomware tools and methods have been proposed [60, 63, 78, 81, 83, 85, 113] and claim to have reasonable detection rates. However, they still fall short at detecting zero-day ransomware attacks, explaining why question of the claimed better performance and the methods often work for given malware family only. Moreover, malware with obfuscated code is often bypassed or falsely detected in those approaches. To overcome the limitations of the current prominent methods, I have proposed an automatic hybrid analysis tool with an explainable component showing the relations at a multi-level with behavioral chains. I analyze the extracted features at DLL, function call, and assembly level using hybrid analysis, association rule mining, and behavioral chain analysis. The prototype AI-powered ransomware detection (AIRaD) tool stands on the proposed architecture and gives the user the flexibility for ease of use with detailed analysis. Furthermore, my method detects zero-day ransomware binaries among a broad range of malware families.

Most ransomware detection-related works are based on either static or dynamic analysis, and few on hybrid analysis, which consists of both. Detection is also done by analyzing network or file system activities. Below I have categorized works into different sub-sections.

2.1 API analysis based detection techniques

A majority of the work is based on API features, or function calls because these are essential in implementing malware functionalities. Here, I discuss some of those notable works.

Takeuchi et al.[113] have proposed ransomware detection using deep inspection of API call sequence and support vector machines as classifiers with 97.48% accuracy and missing rate of 1.64%. They have included the number of occurrences of the given n-gram of API in their vector model. Their proposed feature vector model improved the detection performance with fewer false positives. The model is not tested with other malware families and lacks the explainable component of their chosen machine learning algorithm.

Hampton et al.[78] studied the behavior of ransomware in the Windows system and identified API calls specific to it. The frequency of API usage among ransomware and normal binaries is useful for identifying ransomware without comparing the code signature. They claim that their approach can better understand the ransomware strain in terms of API calls. Handling of obfuscated binaries that uses obscure API calls is missing in this research.

Bae et al.[60] have used the Intel PIN tool to generate windows API call sequences and used the N-gram approach and TF-IDF. They reported accuracies for ransomware, malware, and benign application using six different machine learning classifiers. They have reported the highest accuracy of 98.65% and compared with other works. The test environment runs with an execution time limit of five minutes which malware writers often fool. Steps should have been taken to handle various anti-malware analysis techniques deployed by the adversaries. Their approach could have classified a given malware family instead of a whole set of malware.

Canzanese et al. [63] have analyzed system call traces utilizing the N-gram language model, TF-IDF, and machine learning classifiers (Logistic regression and support vector machines) to detect malicious processes. They have claimed that their proposed system would alarm the user if some unintended behaviors are observed, which includes activities like host modifications. Their study showed a small set of systems call 3-grams providing a comparable detection accuracy to more complex models. Their detection approach is not resilient to the obfuscated behavior of the malware.

Hwang et al. [81] have proposed a two-stage mixed ransomware detection model using the Markov model and Random Forest. They leveraged the Windows API call sequence pattern to build a Markov model, then used the Random Forest machine learning model to the remaining data. The overall achieved accuracy was 97.3% with 4.8% FPR. They performed a dynamic analysis in a sandbox environment and grouped collected APIs into different categories. Their approach could have addressed the handling of obfuscated binaries. What if binaries detect the sandbox environment and do not execute at all? Would the accuracy be improved after adopting a hybrid approach? Answers to such queries are also expected.

Ki et al. [85] have assigned alphabet letters to API functions and apply a DNA sequence alignment algorithm to extract common API call sequence patterns of malicious functions. They claim that the sequence analysis is a better approach as the malware authors can insert dummy and redundant function calls to make the frequency analysis method ineffective. The experimental result showed an accuracy of 99.8%. However, their approach could not handle the detection of obfuscated binaries.

Shaukat et al. [108] have proposed the RansomWall tool, which combines the features obtained through static and dynamic analysis. They monitor file operations dedicated to encryption purposes. They have claimed to detect zero-day ransomware samples. Their approach achieved an accuracy of 98.25% with the Gradient tree boosting algorithm. They have listed some suspicious Windows cryptographic function calls, but extensive analysis of those is missing.

Intending to overcome the limitations of supervised learning algorithms, Sharmeen et al.[107] have proposed a semi-supervised framework to learn unique ransomware behavioral patterns using deep learning techniques. They claim that their model is scalable to accommodate new variants of malware executable. The ransomware samples were run in an isolated environment using a Cuckoo sandbox with varying run times from four to nine minutes. Their approach ignored the samples that could not run in the given experimental settings, making the model less robust. It misses the obfuscated nature of malicious binaries and impacts the detection of obfuscated ransomware samples.

Arabo et al.[56] have done process behavior analysis to determine whether a given sample is ransomware or not. The study starts with which APIs are called and how many system resources are used? The dynamic run result is analyzed to get the feature statistics fed into various supervised and unsupervised machine learning models. The analysis is also done using file extension, API calls made, and disk usage. This particular analysis aims to give users a quick alert if the binary under consideration is a possible ransomware executable. However, their approach has low accuracy and lacks false-positive analysis.

Takeuchi et al.[113] have proposed ransomware detection using deep inspection of API call sequence and support vector machines as classifiers with 97.48% accuracy and missing rate of 1.64%. They have included the number of occurrences of the given n-gram of API in their vector model. Their proposed feature vector model improved the detection performance with fewer false positives. The model is not tested with other malware families and lacks the explainable component of their chosen machine learning algorithm.

2.2 PE file features based detection

Khan et al. [83] have proposed a digital DNA sequencing engine for ransomware detection using Naive Bayes, Decision stump, and AdaBoost algorithms for classification. The features are obtained from the pre-processed data using Multi-Objective Grey Wolf Optimization (MOGWO) and Binary Cuckoo Search (BCS) algorithms. Then the digital DNA sequence is generated for the selected features using the design constraints of DNA sequence and k-mer frequency vector. The experiments show the highest accuracy of 87.9% with AdaBoost. This approach has not mentioned the handling of obfuscated binaries. Moreover, more information about the initial feature, dataset, and methods could have been given. Yuxin et al. [119] have used opcode sequences extracting static features from PE file of malware samples. They have used a deep belief neural network and compared their performance with support vector machines, decision trees, and the k-nearest neighbor algorithm. A deep belief network was used as an autoencoder to extract the feature vectors.

A block cipher algorithm is detected to prevent ransomware infection in work proposed by Kim et al.[86]. The sequence and frequency characteristics are obtained from the opcode of binary files. They have considered Alf and Vegard's RISC (AVR) processor microcontroller for their experiment, where they use a convolutional neural network. Their approach is restricted to static analysis and will not capture the run-time behavior of the malware. The authors have claimed a high accuracy of their system. They could have explained how their approach could handle the anti-analysis behavior of the ransomware samples.

2.3 I/O file system based detection

Continella et al. [68] proposed ShieldFS, which analyzes low-level I/O file system requests. If a write operation is suspicious, it reverts the current process file operations. Billions of I/O requests generated from numerous benign systems were evaluated to design the defense system. They claim that their proposed technique can detect malicious activity like file encryption and successfully recover the original files.

Alam et al.[53] have proposed ransomware prevention via performance counters referred to as RAPPER. They have used neural networks and Fast Fourier Transformation with hardware performance counters (HPCs) as event traces. Perf, a well-known tool in Linux, is used to monitor the HPCs and observe the performance counters and the system behavior. The authors of this work claim this behavior as a good advantage for getting the feature set for their machine learning model. The accuracy of their approach is based on given ransomware samples. Their work could have shown the efficiency of scaling the learning model and the approach to resilience to obfuscation behavior of ransomware samples.

2.4 Defense by encryption tracking

PayBreak [87] tried to recover the data corrupted by ransomware by extracting the encryption key. This approach only works for symmetric encryption or hybrid encryption. Here, no accuracy evaluation is done. Moreover, they have not explained how they would handle the false positives as encryption behavior can be found in normal user profiles. Kharaz et al. [84] proposed a ransomware detection framework called UNVEIL, which tracks the encryption behavior of ransomware. They tracked IO operations calculating entropy scores. The authors reported the detection of 13,000 malware samples across different malware families. However, they did not verify the accuracy of their system.

2.5 Detection in mobile platform

Andronio et al. [55] proposed the HELDROID framework for detecting ransomware. The framework works by searching for the requisite ransomware elements within mobile applications. It can detect whether an app is trying to encrypt or lock the device without permission from the user. The results of testing HELDROID on APKs comprising ransomware, goodware, scareware, and malware showed almost zero false positives.

Faghihi et al.[72] have presented a data-centric detection and mitigation against smartphone crypto-ransomware using dynamic analysis and hash-based techniques. They analyze the user's data along with its entropy values to make detection decisions. API calls related to file operations are intercepted by function hooking techniques. Then entropy and structure of data are monitored to detect and neutralize the ransomware. Their approach claims a high accuracy with a low false-positive rate but shows a low resilience to obfuscation behavior of ransomware executable.

2.6 Network based detection

Rafal et al. [90] have used distributed machine learning for Botnet activity detection. They proposed using and implementing cost-sensitive distributed machine learning through distributed Extreme Learning Machines (ELM), distributed random forest, and distributed random boosted-trees. Data were analyzed in NetFlow (ports, protocols, IPs, packets, etc., were considered). To make the analysis efficient and scalable, they have proposed to collect the NetFlow data in an HDFS system and Map-Reduce. Experiments were run using 1 and 8 Apache spark nodes. Using spark clusters, there were able to improve the training process by a factor of 4.5 for ELM and 3.7 for Random forest and gradient boosting trees. Overall classification efficiency was considered better for ELM than others for various scenarios, so it is proposed as an efficient to use.

2.7 APT attacks and detection

An advanced persistent threat(APT) is a type of cyberattack where an attacker uses sophisticated techniques to gain unauthorized access to a system or a network [7]. The attack remains undetected for days to months or even a few years. The advanced part of APT refers to using multiple advanced tools and techniques to find a vulnerability to conduct an attack. The persistent refers to the long-term access to the target system. The threat is the potential adverse action by the bad actors.

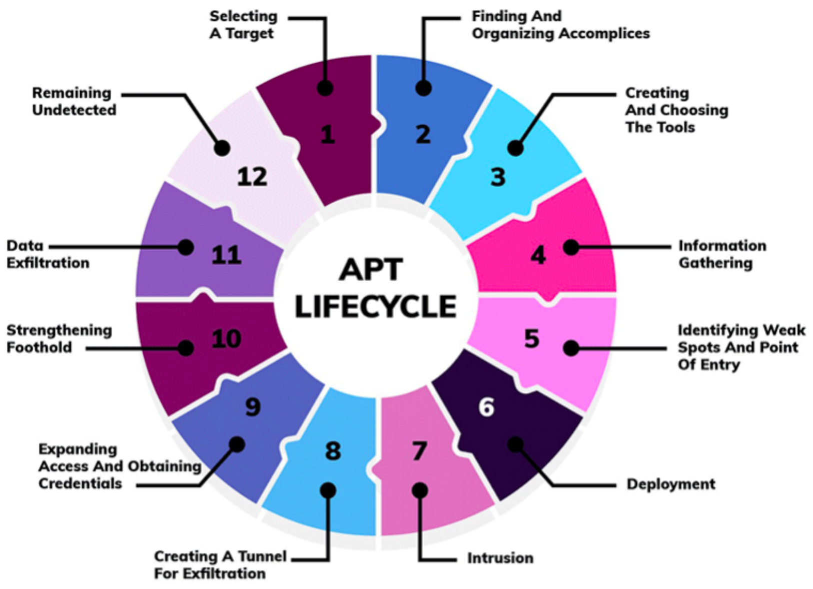

APT LIFECYCLE
1 Selecting A Target
2 Finding And Organizing Accomplices
3 Creating And Choosing The Tools
4 Information Gathering
5 Identifying Weak Spots And Point Of Entry
6 Deployment
7 Intrusion
8 Creating A Tunnel For Exfiltration
9 Expanding Access And Obtaining Credentials
10 Strengthening Foothold
11 Data Exfiltration
12 Remaining Undetected

Figure 10: Advanced persistent threat life-cycle [7]

Figure 10 shows the detailed steps involved in an APT attack. The first step is reconnaissance, where the attacker identifies who the target is and plans for the attack. Then the APT actors try to access the target network by using malicious attachments, spear-phishing campaigns, exploiting vulnerabilities, and other similar techniques. Once they access the target network, they inject malware and create backdoors that allow for unauthorized remote access. The next stage is a lateral movement by detecting additional vulnerabilities and installing more backdoors. The APT actors also try to create a remote tunnel for data transfer at a later stage. Data exploration is the next step where the attackers locate data of interest to them. The next stage is the compromise step, where data exfiltration occurs. The data transfer occurs from the victim's network to the attacker's server via a remote tunnel. The final step is to remove the footprints of the APT attack so that the victim may not know that their network is compromised, but still, APT actors try to leave some undetected footprints for future attacks.

The steps till data exfiltration of APT attack are generally similar to the ransomware attack plan. The difference is that in a ransomware attack, the data is encrypted, and still, there is possible double extortion where the attacker may encrypt the victim's file and sell the data to the dark web.

Ghafir et al. [76] have proposed a machine learning-based APT attack detection framework. The framework mainly focuses on different methods of threat detection using real network traffic. They try to establish correlation among the output of detection methods, and finally, attack prediction is made, which can fire early alerts. They claim an accuracy of 84.8%. However, their approach has to be polished to deal with false positives and missed detection.

Li et al. [92] have used a hierarchical approach for APT detection using attention-based Graph Neural Networks. They claim that previous strategies that use coarse-grained correlation graphs cannot explore log node attributes and mainly depend on system calls. The authors claim to capture the features at both system and network levels.

3 Proposed Research

The proposed methodology is a ransomware detection framework via deep inspection of its multi-level features leveraging hybrid analysis, advanced reverse engineering, and Artificial Intelligence (AI) techniques. This research aims at proposing an AI-based ransomware analysis and detection framework using a combination of both static and dynamic malware analysis techniques. Specifically, I proposed a deep inspection approach for multi-level profiling of crypto-ransomware, which captures the distinct features at DLL, function call, and assembly levels. I have shown how the code segments are correlated at these levels for studied samples. This hybrid multi-level analysis approach includes advanced static and dynamic methods and a novel strategy of analyzing behavioral chains with AI techniques. Moreover, association rule mining, natural language processing techniques, and machine learning classifiers are integrated for building ransomware validation and detection model. I experimented with 550 crypto-ransomware samples (collected from VirusTotal), and the result exhibited that multi-level profiling can better detect ransomware samples with higher accuracy. The multi-level feature sequence can be extracted from most applications running in the different operating systems; therefore, this method can detect ransomware for devices on multiple platforms. Below I have discussed about the goal and objectives.

This research aims to design and implement an advanced malware detection framework called "AI-Powered Ransomware Detection Framework." To achieve this goal, the following objectives identified need to be fulfilled.

Objective 1: Malware sample collection and categorization.

Objective 2: Design an advanced reverse engineering framework for pre-processing and feature extraction.

Objective 3: Design a machine learning training, validation, and testing model for malware detection.

Objective 4: Implement components of the proposed framework.

Following tasks need to be performed to fulfill the objectives mentioned above.

- Collect malware samples from various sources such as Virustotal and open-source malware repository theZoo [43].
- Create a virtual work environment for experimentation using Vmware with host and server machine.
- Design and implement a pre-processing component using objdump of Linux and open source Portable Executable(PE) parser tool.
- Design and implement the extractor components of the proposed framework for feature analysis.
- Implement NLP techniques such as n-gram probability, term-frequency, and TF-IDF for feature generation and analysis.
- Implement dynamic binary instrumentation using PIN tool.
- Implement various supervised machine learning techniques for training and testing the ML model.
- Compare the performance of the proposed technique with the other related works and products.
- Perform behavioral analysis of malware chains.
- Build detection signatures based on association rules and behavioral chaining.
- Implement the multi-level framework for ransomware detection.

The following chapters will show how I progressed from one method or approach to another and, finally, a polished proposed framework.

4 Static Analysis based Machine Learning Framework for Ransomware Detection

In this work, I developed a reverse engineering framework incorporating feature generation engines and machine learning (ML) to detect ransomware efficiently. This framework is used to perform multi-level analysis (such as raw binaries, assembly codes, libraries, and function calls) to examine better and interpret the purpose of malware code segments. I leverage the object-code dump tool (Linux) and portable executable (PE) parser to decode binaries to assembly level instructions and dynamic link libraries (DLLs). Both ransomware and normal binaries are considered to conduct experiments where samples are first pre-processed to extract features. Then different (supervised) ML techniques are applied to classify these samples. Experimental results reported the performance, i.e., the detection accuracy of ransomware samples which varied from 76% to 97% based on the ML technique used. In particular, among the eight ML classifiers tested, seven of these performed well with a detection rate of at least 90%. This study also demonstrated that the combination of static level analysis at the ASM level and Dll-level could better distinguish ransomware from normal binaries.

4.1 Proposed Methodology

The proposed ransomware detection framework has two major components: Feature Generation Engine and Machine Learning Model as shown in Figure 27. The Feature Generation Engine is based on feature generation from ransomware and normal binaries using reverse engineering and pre-processing (described in Section 4.1.1). Ransomware and benign application samples are each fed to the reverse engineering process, where the disassembler extracts the information from the binaries. This output is pre-processed and sent to the extractor component (described

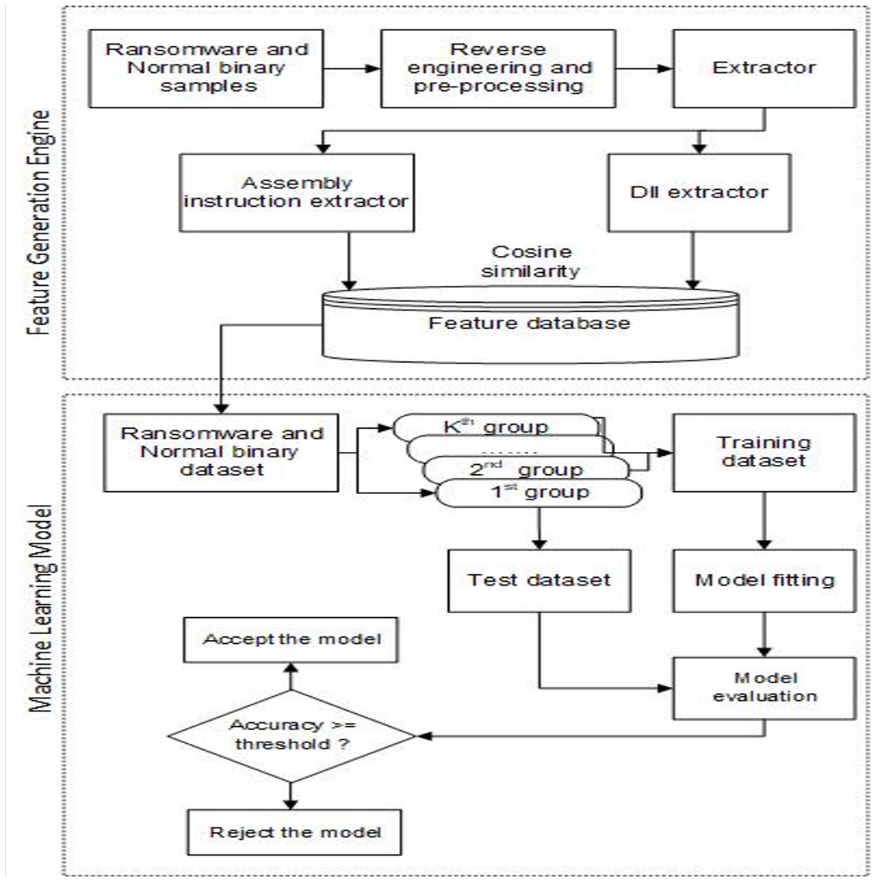


Figure 11: Framework for feature extraction and detection of ransomware using machine learning

in Section 4.1.2), which parses the processed object file and extracts the dlls and assembly instruction set. The extractor program then does the frequency count and builds the feature database. The similarity between binaries is found using cosine similarity (described in Section 4.1.3). From the Feature Generation Engine, I get the ransomware and benign binaries dataset, which is the starting component for the machine learning model. I perform resampling using K-fold cross-validation (described in Sections 4.1.5 and 4.1.6). The dataset is trained and tested using different supervised machine learning classifiers (explained in Section 4.1.7). These classifiers are evaluated based on the ransomware detection rate (described in Section 4.1.8).

4.1.1 Reverse Engineering and pre-processing

Reverse Engineering [74] is taking a binary file that is meant to be read by the computer and using the opcodes to generate assembly, and then reading that assembly to help accomplish whatever goals we may have. I performed the reverse engineering using static analysis of the ransomware and normal binaries leveraging the existing disassembler objdump [112] and PE parser [31].

I followed the Intel syntax to disassemble and analyze the binaries. The objdump program, also known as assemblydumper, outputs the assembly-level code segment containing assembly instructions. The PE parser is used to extract the code segment containing dlls required to execute the PE file. The pre-processor module scans and processes different code segments generated by the assemblydumper and the PE parser.

4.1.2 Extractor

I modeled an extractor tool that is required for mining the binary dump file. It has two basic components as Assembly instruction extractor and Dll extractor. The first component parses the output of the objdump program. It then builds the frequency table of all the assembly instructions. The second component parses the output of the PE parser and creates the frequency table of all the dlls used. The frequency distribution for assembly instructions and dlls are stored in the feature database, which is a MySQL database.

4.1.3 Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of n dimensions. It's value ranges from zero to one. If two vectors have the same orientation, then the cosine similarity becomes one. Whereas, if the orientation is perpendicular to one another, then it becomes zero. Given two vectors of attributes P and

Q, the cosine similarity is given as:

$$cos(\theta) = \frac{P.Q}{|P||Q|} = \frac{\sum_{i=1}^{n} P_i.Q_i}{\sqrt{\sum_{i=1}^{n} P_i^2}\sqrt{\sum_{i=1}^{n} Q_i^2}} \tag{1}$$

Here, P and Q are different ransomware or normal binaries. Pi and Qi are the corresponding features of P and Q, respectively. I measure the cosine similarity at instructions and Dll level for ransomware binaries.

4.1.4 Machine Learning Model

From the Feature Generation Engine, I get the ransomware and benign binaries dataset which is the starting component of our machine learning training model as shown in Figure 27. This training model accepts only those machine learning classifiers with ransomware detection accuracy greater than the threshold value. The expert user can set this threshold. Here, for our experiment, I set it as 90%. The resampling of the dataset is done with k-fold cross-validation, which is described below sections.

4.1.5 Resampling

Resampling methods involve repeatedly drawing samples from a training set and refitting a model of interest on each sample to obtain additional information about the fitted model [82]. For example, we can repeatedly draw samples from the training dataset and estimate the variability fit of a Random Forest or any other machine learning model. There are different approaches for resampling, but I choose k-fold cross-validation for my experiment due to its good performance and broader acceptance.

4.1.6 K-fold Cross-Validation

K-fold cross-validation is a statistical method to compare and select a model for a given predictive modeling problem. It is a procedure used to estimate the performance or accuracy of the machine learning model on new and unseen data. This technique uses the well-known parameter K, which is the number of groups that a given dataset is split into. It involves randomly dividing the set of observations into K groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining *K-1* folds [82]. If the value of K is chosen with proper care and understanding, then this validation method results in a less biased estimate of the machine learning model than due to other methods such as Leave-one-out cross-validation and training and testing set. When $K = n$, where n is the size of the dataset, then K-fold cross-validation becomes leave-one-out cross-validation. In the experiment I performed K-fold cross-validation using $K = 10$. This value has been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor very high variance [82]. The results of our experiment also prove this claim.

4.1.7 Supervised Machine Learning Classifiers

Machine learning usage is becoming a more widespread technique for ransomware detection. Here, I have applied various supervised machine learning algorithms to ransomware-labeled datasets. In particular, I use Bayesian Network, Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, and Adaboosting with different classifiers. These classifiers were implemented using Weka [48] version 3.8.2. A brief discussion of the machine learning classifiers that I have used follows below.

4.1.8 Model Fitting and Evaluation

The training and testing dataset is generated using the K-fold cross-validation technique as defined in the earlier subsection. The model fitting is done for each supervised machine learning classifier using the training dataset, and its accuracy is evaluated using the test dataset. The classifier model is accepted if the accuracy is equal to or greater than the defined threshold value.

4.2 Experiments and Results

In this section, I explain data collection, experimental protocols, evaluation measures, and experimental results. I performed experiments on different types of datasets depending upon the type of feature used.

4.2.1 Dataset

A total of 302 samples of malware was collected from various sources such as Virus Total, Virus Share, and open-source malware repository theZoo [44]. Our challenge was to categorize and confirm whether each malware is ransomware or not.
I developed a CategorizerTool that leverages RESTful API provided by VirusTotal [46] which uses two parameters. The *resource* parameter is the hash value of the malware, whereas the *apikey* parameter is specific to the user account. VirusTotal scans the provided *resource* using different anti-virus engines and outputs various statistics, including the malware type. More than fifty different anti-virus engines classify each binary as either ransomware, malware, or another type. Also, each engine has different class names for the binaries and is regularly updated. To have consistency in categorizing ransomware binaries, I choose Malwarebytes[27] and store the results of our CategorizerTool in the *MySQL* database. The malware binaries with no classification result were ignored as not being a ransomware sample.

Based on the classification done in June 2018, I have the family groupings as shown in Table 2.

Table 2: Ransomware families

Family Name	No. of Samples
Locky	74
Teslacrypt	60
FileLocker	17
FileCryptor	5
Troldesh	4
Cryptowall	4
Torrentlocker	4
CryptoLocker	3
ZeroLocker	2
CryptoTorLocker	2
CTBLocker	1
XORIST	1
WannaCrypt	1

The total number of ransomware of different families is found to be 178. I take 178 number of benign executables. The normal executables include samples from Windows 10 operating system and open source applications. It includes normal to advance programs such as *browser.exe*, *process.exe*, *network.exe*, *bitlocker.exe*, *puttygen.exe*, *ssh-keyscan.exe*, *FileZilla_ Server.exe*, *sshsecureshellclient.exe*, *WinScp.exe*, *OpenSSHClient.exe* and so on. The *FileZilla*, *WinScp*, and *BitLocker* for example uses cryptographic and communication operations which resembles some of the functionalities of ransomware. I choose normal binaries of size in the range of 110 KB to 10 MB so as to resemble the size of ransomware samples. The normalized size distribution of ten ransomware families and normal binaries is shown in Figure 12.

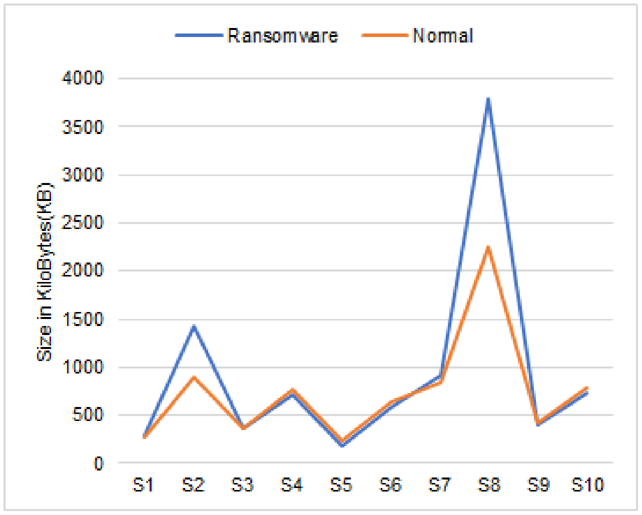


Figure 12: Comparison of file size among ransomware and normal binaries

4.2.2 Experimental Protocols and Evaluation Measures

The collected samples are given as input to the feature extraction engine and then output to the machine learning training model. This whole experiment is conducted in a machine with the following configuration: Intel(R) Core(TM) i7-5500U CPU @ 2.40 GHz 2.39 GHz, 8.00 GB RAM, and 1 TB disk space.

I evaluated the different supervised machine learning algorithms by various performance metrics listed below. I used the confusion matrix of a classifier to calculate these metrics. It tells how often the classifier is correct.

$$True\ positive\ rate\ (TPR) = \frac{TP}{TP + FN} \tag{2}$$

$$False\ positive\ rate\ (FPR) = \frac{FP}{FP + TN} \tag{3}$$

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{6}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

In the above equations, TP is a true positive, representing the number of ransomware samples correctly classified. TN is a true negative, which means the number of normal samples correctly classified. FP is a false positive which represents normal binaries incorrectly classified as ransomware. FN is a false negative which represents ransomware incorrectly classified as normal binary. TPR gives the value of predicted ransomware classified correctly as ransomware, whereas FPR gives the value of normal binaries incorrectly classified as ransomware. Precision defines the accuracy of the machine learning model in terms of classifying the relevant instances. Recall establishes the ability to find the relevant instances in the dataset. F-measure is the harmonic mean of precision and recall and estimates the performance of the given machine learning model. The accuracy is defined by the ratio of correctly predicted instances to the total testing instances expressed in percentage.

4.2.3 Experimental Results

Here, I provide an experimental evaluation of the proposed ransomware detection system. I experimented with assembly level instruction, Dll level, the combination of the above two, and the result is analyzed accordingly.

Table 3: Machine learning algorithms evaluation for assembly level instructions

Machine learning Classifier	TPR	FPR	Precision	Recall	F-measure	Accuracy (%)
BN	0.961	0.039	0.961	0.961	0.961	96.0843
LR	0.795	0.205	0.795	0.795	0.795	79.5181
SMO LK	0.973	0.028	0.974	0.973	0.973	97.2892
SMO PK	0.934	0.067	0.934	0.934	0.934	93.3735
J48	0.967	0.033	0.967	0.967	0.967	96.6867
RF	0.976	0.025	0.977	0.976	0.976	97.5904
Ada J48	0.973	0.027	0.973	0.973	0.973	97.2892
Ada RF	0.979	0.021	0.979	0.979	0.979	97.8916

Table 4: Machine learning algorithms evaluation for DLL level

Machine learning Classifier	TPR	FPR	Precision	Recall	F-measure	Accuracy (%)
BN	0.766	0.246	0.825	0.766	0.752	76.5579
LR	0.819	0.182	0.819	0.819	0.819	81.8991
SMO LK	0.872	0.123	0.886	0.872	0.872	87.2404
SMO PK	0.872	0.122	0.892	0.872	0.871	87.2404
J48	0.875	0.122	0.881	0.875	0.875	87.5371
RF	0.902	0.095	0.908	0.902	0.902	90.2077
Ada J48	0.896	0.100	0.904	0.896	0.896	89.6142
Ada RF	0.908	0.089	0.912	0.908	0.908	90.8012

4.2.3.1 Accuracy at Assembly Level Instruction

The experiment is done with the assembly instruction dataset consisting of 599 unique instructions. The results in Table 3 clearly show that the detection rate for ransomware is above 90% in seven classifiers among eight of them. The accuracy is more than 97% for Random forest, AdaboostM1 with J48, and AdaboostM1 with RF. The mean absolute error is also less than 0.05 in these classifiers. We can observe the least TPR of 0.795 for just one algorithm, i.e., logistic regression, but for other machine learning classifiers, the TPR is significantly higher above 0.934 and up to 0.979. The FPR is minimum for classifiers other than Logistic regression. The average accuracy for assembly level instruction analysis is found to be 94.46%.

Table 5: Machine learning algorithms evaluation for combined Assembly instructions and DLLs dataset

	BN	LR	SMO LK	SMO PK	J48	RF	Ada J48	Ada RF
Accuracy	97.076	89.1813	96.7836	90.6433	97.076	97.9532	97.6608	97.6608

Table 6: Ransomware detection accuracy for individual families

	BN	LR	SMO LK	SMO PK	J48	RF	Ada J48	Ada RF
Locky	97.2414	92.4138	95.8621	92.4138	97.931	96.5517	97.931	98.6207
TeslaCrypt	95.7627	91.5254	89.8305	88.1356	95.7627	97.4576	97.4576	97.4576

4.2.3.2 Accuracy at Dll Level

The experiment is conducted with the Dll dataset consisting of 350 unique dlls. The results in Table 4 shows the minimum detection accuracy of 81.89% for Logistic Regression. It is a maximum of approximately 90% for Random Forest and AdaboostM1 with RF. Adaboost generally improves the detection accuracy significantly for weak classifiers such as Decision stump. I experimented with this and found an improvement from 77.74% to 85.45%, i.e., with Decision stump, it was only 77.74%, but it improved to 85.45% when using AdaboostM1 with Decision stump. The performance did not improve significantly in the case of J48 and Random forest. The mean absolute error is least at 0.1307 for AdaboostM1 with RF. The average accuracy for the Dll level analysis is found to be 86.38%.

4.2.3.3 Accuracy at Combined Level

In this experiment, I combined all the unique assembly instructions and dlls to form a single feature set with 949 features for each sample. The accuracy is listed in Table 5, and its comparison with another type of datasets is shown in Figure 13. It is easy to see that the detection rate improves with the combined feature set for each sample into consideration. For instance, with a combined feature set, the average accuracy is 95.5% which is approximately 1% more than the average detection rate with an assembly instruction dataset.

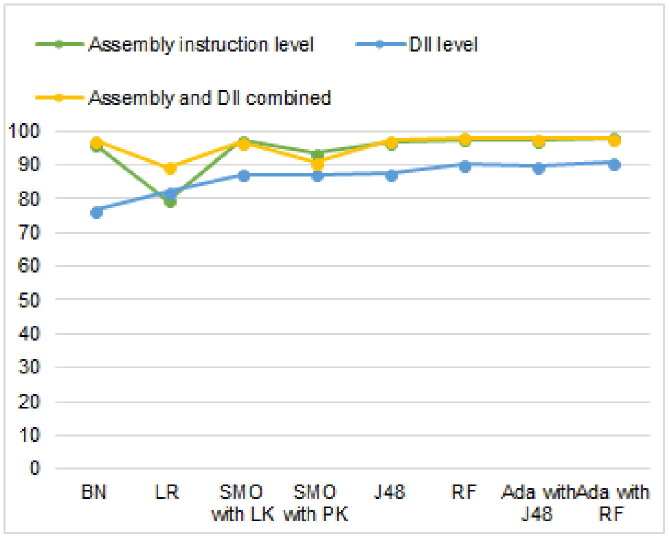


Figure 13: Comparison of detection accuracy (in percentage) using individual and combined features of assembly instructions and dlls

4.2.3.4 Individual Family Detection Accuracy

In this experiment, I took a separate combined feature set for the Locky and TeslaCrypt ransomware family. Locky ransomware dataset consists of 822 unique features while it was 782 for TeslaCrypt. Table 6 shows the accuracy rate for each machine learning classifier for each family. The detection accuracy is higher for the Locky family than the TeslaCrypt family, with the maximum up to 98.62% for AdaboosM1 with RF.

4.3 Model Building Analysis

Machine learning classifiers build the learning model to classify the new unknown samples. I compared the model building time (expressed in seconds) for different classifiers for three types of a dataset whose result is shown in Figure 14.

The result demonstrates that the model building time increases with the number of features. The combined dataset having 949 unique feature set has longer building time for each machine learning classifier compared to other datasets.

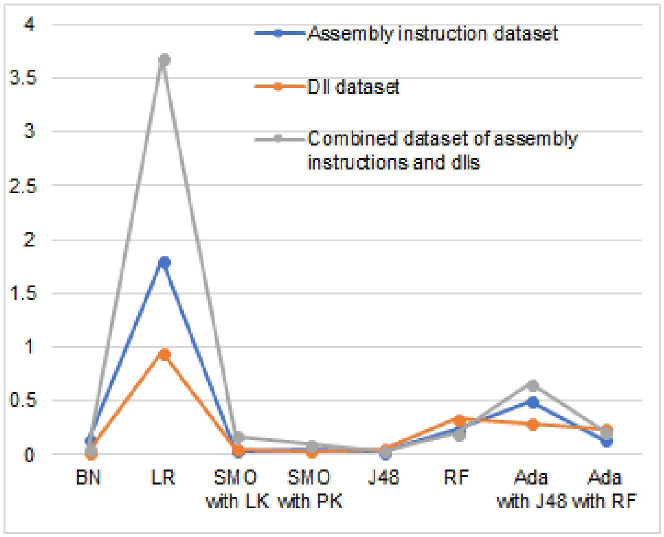
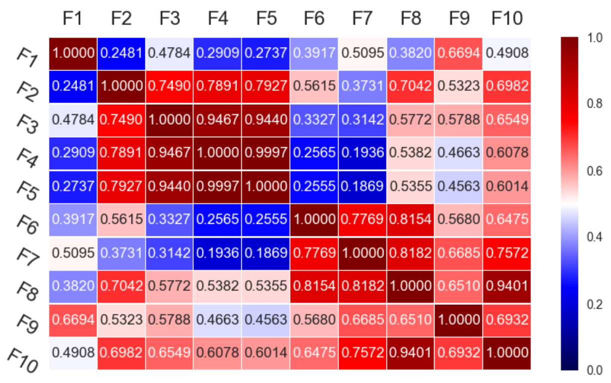


Figure 14: Comparison of model building time (in seconds) of different machine learning classifiers for individual and combined features dataset

4.4 Cosine Similarity Analysis

Similarity among ten different ransomware families is found using cosine similarity. I use heatmap as shown in Figure 15 to visualize the similarity. Similarity score ranging from 0 to 1 is shown in the right portion, whereas the top and left portion shows the ransomware families. The Figure 15 shows that ransomware families *CryptoLocker(F1)*, *CryptoTorLocker(F2)*, *CryptoWall(F3)*, *Locky(F6)*, *TeslaCrypt(F7)*, *TorrentLocker(F8)*, *TrolDesh(F9)*, and *WannaCrypt(F10)* are similar with similarity score of at least 0.1869. *FileCryptor(F4)* and *FileLocker(F5)* are almost

identical with the high similarity score of 0.9997. These two samples have unique hash values but similar characteristics though they belong to different ransomware families. The average similarity score is 0.571, and the standard deviation is 0.213. These two statistical metrics can be used to create malware signatures. Any new sample with an average similarity score greater than 0.5 can be categorized as being ransomware. Thus, heatmap analysis provides the visual representation of similarity



	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
F1	1.0000	0.2481	0.4784	0.2909	0.2737	0.3917	0.5095	0.3820	0.6694	0.4908
F2	0.2481	1.0000	0.7490	0.7891	0.7927	0.5615	0.3731	0.7042	0.5323	0.6982
F3	0.4784	0.7490	1.0000	0.9467	0.9440	0.3327	0.3142	0.5772	0.5788	0.6549
F4	0.2909	0.7891	0.9467	1.0000	0.9997	0.2565	0.1936	0.5382	0.4663	0.6078
F5	0.2737	0.7927	0.9440	0.9997	1.0000	0.2555	0.1869	0.5355	0.4563	0.6014
F6	0.3917	0.5615	0.3327	0.2565	0.2555	1.0000	0.7769	0.8154	0.5680	0.6475
F7	0.5095	0.3731	0.3142	0.1936	0.1869	0.7769	1.0000	0.8182	0.6685	0.7572
F8	0.3820	0.7042	0.5772	0.5382	0.5355	0.8154	0.8182	1.0000	0.6510	0.9401
F9	0.6694	0.5323	0.5788	0.4663	0.4563	0.5680	0.6685	0.6510	1.0000	0.6932
F10	0.4908	0.6982	0.6549	0.6078	0.6014	0.6475	0.7572	0.9401	0.6932	1.0000

Figure 15: Heatmap for Similarity Score among Ransomware Family

scores and the method to create executable signatures.

4.5 Summary

In this work, I proposed a ransomware detection framework leveraging reverse engineering, static analysis, and machine learning. I built the feature database from analysis at assembly and Dll level of binaries and trained and tested our machine learning training model. The experimental results show that among eight supervised machine learning classifiers, our framework could achieve an accuracy of more than 90% (96.5% on average) for seven of them for the combined feature dataset and assembly level instruction dataset. The ransomware detection rate is significantly higher for the dataset with an integrated feature dataset with a minimum of 89.18% for Logistic regression and a maximum of 97.95% for the Random Forest. From this work, I claim that static analysis of binaries at the assembly and Dll level is crucial to building distinguishing characteristics for ransomware detection using machine learning.

5 A Multi-Level Ransomware Detection Framework using Natural Language Processing and Machine Learning

In this work, I proposed a multi-level big data mining framework combining Reverse engineering, Natural Language Processing(NLP), and Machine Learning(ML) approaches. The framework analyzes the ransomware at different levels (i.e., Dynamic-link library, function call, and assembly instruction level) via different supervised ML algorithms. Apache Spark was employed for the faster processing of large generated feature sets. Portable Executable (PE) parser and Objectdump tool of Linux system were used to get the raw data from the ransomware and normal binaries processed further using our custom-built NLP processing. The n-gram probabilities, term-frequency, and inverse document frequency (TF-IDF) were used to generate the final feature sets. Experiments were performed with different 'N' values of the n-gram language model that shows that the ransomware detection accuracy is inversely proportional to the value of N. Among the five chosen supervised classifiers, Logistic regression outperformed others with a detection rate of 98.59% for generated TF-IDFs trigrams at combined multi-level, which is an improved accuracy compared to individual levels.

5.1 Proposed Methodology

The proposed methodology is a multi-level ransomware detection framework comprising six major components: DLL tracker, Function call tracker, Assembly instruction tracker, Detector engine, Action engine, and Passive analyzer. This multi-level framework is run in an active mode to analyze the given binaries at three levels, as shown in Figure 16. It is initiated with the detection counter (dc) set to zero. This framework tracks the detection rate at each level going from DLL to the assembly instruction level, so it is named a multi-level framework. At level 1, the

DLL tracker interacts with the detector engine, moves to the second level with the function call tracker, and then finally moves to the assembly instruction tracker. The details of each major component are described below sections.

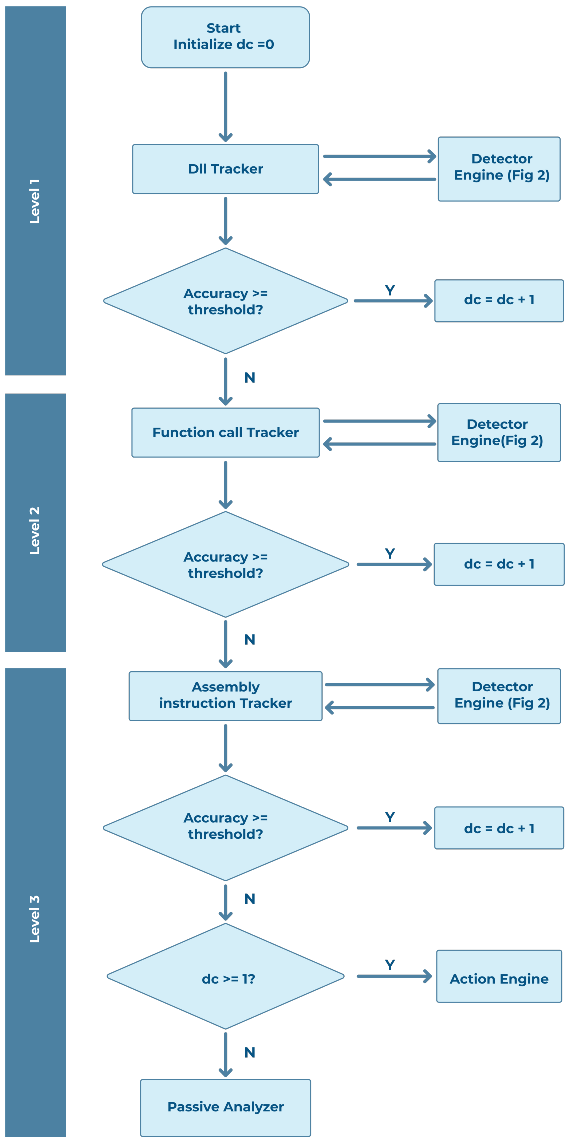


Figure 16: Multi-level ransomware detection framework

5.1.1 DLL tracker

DLL tracker analyzes the DLLs of a given binary using the detection engine, as shown in Figure 16, and calculates its classification accuracy. The details of detector engine framework is explained in Section 5.2. The detection counter is incremented by one if the accuracy is greater or equal to the defined threshold value. The threshold value is set by the expert user or the security team. For the experiment, I considered the threshold as 80%.

5.1.2 Function call tracker

A function call tracker analyzes the function calls of a given binary. It also uses the detector engine and calculates the classification accuracy. The detection counter is incremented if the accuracy obtained is greater or equal to the defined threshold value.

5.1.3 Assembly instruction tracker

An assembly instruction tracker works similarly to DLL and function call trackers. The difference is that the detection counter's value is evaluated here. If that value is greater or equal to one, then the action engine is triggered; else, the passive analyzer comes into play.

5.1.4 Detector engine

A detector engine is a vital component of the proposed multi-level framework. It consists of reverse engineering, pre-processing, big data analysis, natural language processing methods, and machine learning classifiers. Each tracker, at each level, uses this engine to identify whether the given executable is benign or ransomware. This engine is explained in detail with a block diagram in Section 5.2.

5.1.5 Action engine

An action engine is responsible for incident handling and response. When the detection counter's value is greater or equal to one, then the action engine analyzes its further action and alerts the user or system about the detection. Immediate action or preventative actions are implemented via either manual or automatic inspectors. The details of the action engine are beyond the scope of this work. However, more related details can be found on these references [87, 105, 36, 50].

5.1.6 Passive analyzer

When an action engine excludes an executable or binary file, the system monitors using a passive analyzer. The passive analyzer generates the signature of the binary and updates its detection database. The security admin may further escalate the analysis of a particular binary using behavior analyzer techniques such as system monitoring, file access analyzers, and so on. Digging into the details of the passive analyzer is, again, out of the scope of this work. More related information can be read from these references [77, 111, 96].

5.2 Workflow of detector engine

The detector engine works in two phases: Feature generation and Machine learning prediction, as shown in Figure 17. Each phase conducts various operations, which are described in sections given below.

5.2.1 Reverse engineering and pre-processor

Reverse engineering deconstructs a binary executable to generate the assembly op-codes and analyze them to fulfill some meaningful objectives. The life cycle of a binary executable is shown in Figure 18.

The program source code, written either in C or another programming language, is compiled, which involves steps from a lexical analyzer to a code optimizer. The object files generated are linked to a binary file. The loader consists of OS loaders and dynamic link libraries, which finally resolve the code's references to become a

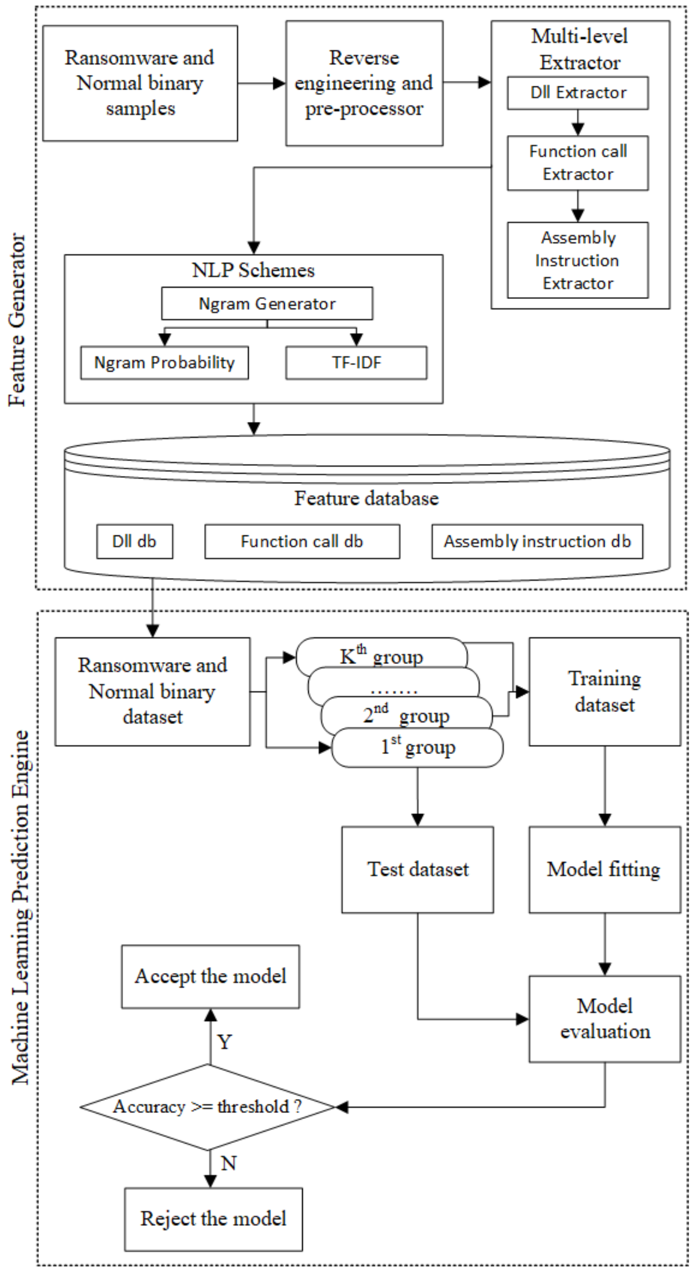


Figure 17: Workflow of detector engine

running executable. The reverse engineering process aims to get the source code functionality as close as possible. I reverse-engineered the ransomware and normal executable using the PE parser tool [30] and Objdump Disassembler.

The PE parser tool is used to get the DLLs and function calls used by the ransomware and normal samples. In contrast, the objdump tool is used to get the

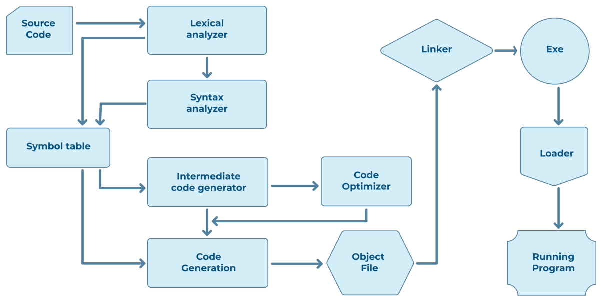


Figure 18: Life cycle of a binary file

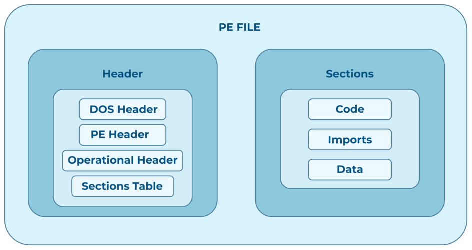


Figure 19: PE file format

assembly instructions associated with each executable sample. The pre-processor component processes the code segments generated by the PE parser and objdump tool.

In this work, I deal with the windows portable executable files. PE file format is a data structure that holds the information that is required for the Windows operating system loader to handle the program code [94]. It is used by windows executable, object code, and DLLs.

PE file consists of PE header and PE sections, as shown in Figure 19.

The first segment is the header, and it contains information about the code and its application type, necessary library or kernel functions, and how much space is needed. The section is the second segment and includes code, import, and data. The text section contains the instructions that the central processing unit executes, and it is the only section that provides for code. More details of PE file format is discussed in "PEFile analysis: a static approach to ransomware analysis" [101].

5.2.2 Multi-level Extractor

The multi-level extractor tool collects the DLLs, function calls, and assembly instructions used in a sequence for a given sample from the processed data of the preprocessor. Below is a brief explanation of each extractor type.

5.2.2.1 DLL Extractor

A dynamic link library referred to as a DLL is a library that contains code and data that can be used by more than one program at the same time. The main benefit of DLL is code re-usability and efficient memory usage. DLL can be user defined or entity/Microsoft defined as shown in Figure 21.

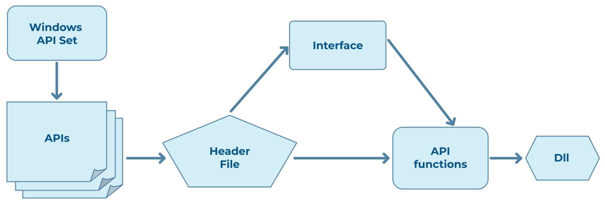


Figure 20: Hierarchy of windows DLL

Figure 20 shows the hierarchy of windows DLL. The Windows API set is the superset that consists of one or more Application programming interfaces(APIs). Each API is a header file with or without interfaces that consists of API functions. DLL makes these API functions act upon and can be considered a bridge between the user space and the kernel space. The DLL extractor component parses the output of the PE parser and lists the DLLs used by the given executable.

5.2.2.2 Function call Extractor

A function call is a piece of code that actually has lines of instructions that makes an impact to the system or user. Each DLL which is implicitly or explicitly linked consists of both import and export functions as shown in the Figure 21.

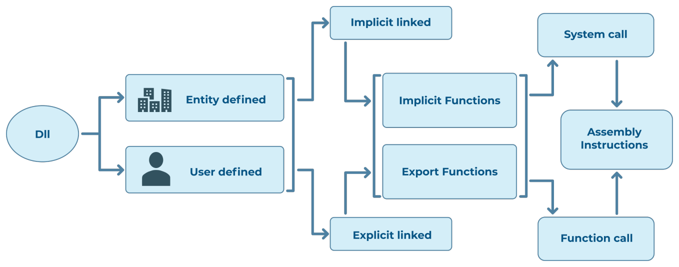


Figure 21: Hierarchy of function calls and assembly instructions in a DLL

Each of those functions consists of several function calls and system calls. While disassembling the binaries, function calls and system calls appear together in sequence. System calls are considered a particular type of function call, so the latter is often used as a common term. In the experiment, I use the function call sequences, which consists of system calls as well.

5.2.2.3 Assembly Instruction Extractor

Assembly instruction is a low-level machine instruction, which is also called machine code. It can be directly executed by a computer's central processing unit(CPU). Each assembly instruction causes a CPU to perform a specific task, like add, subtract, jump, xor, etc. Each function call or system call is implemented via assembly instructions as shown in the hierarchy in Figure 21.

5.2.3 NLP Schemes

NLP language models have proved helpful in recommendation systems, text classification, speech recognition, and so on. I have exploited some popular concepts in this work but applied them to a unique problem domain of the multi-level analysis model of ransomware detection. In this work, NLP schemes are composed of three methods: N-gram generation, N-gram probability, and TF-IDF, which are described below.

5.2.3.1 N-gram Generator

An n-gram model is a type of probabilistic language model that predicts the next possible item in a sequence. N-gram is a contiguous sequence of n items from a given sample of text corpus or speech corpus. In our experiment, it is the DLL, function call, and assembly instruction corpus. The items can be phonemes, letters, words, DLLs, functions call, opcodes, or base pairs depending upon the type of application considered. The value of N can be 1,2,3,4, or any other positive integer. The value for N depends upon the problem domain, type, and nature of the dataset. In text processing tasks, a smaller N usually decreases the classification performance heavily, while the larger N is not considered too relevant. In the experiment, I choose different values of N ranging from 2 to 6 to analyze the detection accuracy of ransomware.
The n-gram generation component is responsible for generating possible sequences of n-grams for a given value of N. I consider only the unique set of n-gram sequences.

5.2.3.2 N-gram Probability scoring

N-gram Probability scoring component is fed with the n-gram sequences obtained from the n-gram generation component. I apply the Markov assumption by considering only the immediate N-1 words.
In a n-gram, we consider the length $n-1$

$$p(w_i|w_1,\ldots,w_{i-1}) = p(w_i|w_{i-n+1},\ldots,w_{i-1})$$

- unigram: $p(w_i)$
- bigram: $p(w_i|w_{i-1})$ (Markov process)
- trigram: $p(w_i|w_{i-2},w_{i-1})$

We can estimate n-gram probabilities by counting relative frequency on a training corpus.

$$\hat{p}(w_b|w_a) = \frac{c(w_a, w_b)}{c(w_a)}$$

N is the total number of words in the training set and $c(\cdot)$ denotes count of the word or word sequence in the training data.

For our experiment dataset, the n-gram sequence will be the DLL, function call, and assembly instruction sequences, and N will be their corresponding total number of sequences in the corpus. The probability scores for each n-gram sequence are stored in a feature database.

5.2.3.3 TF-IDF

TF-IDF is the product of Term frequency(TF) and Inverse document frequency(IDF). Term frequency is simply the number of occurrences of particular n-gram sequences in a binary sample, whereas the IDF is given as:

$$IDF(ngram) = log_e \frac{Total\ no\ of\ binaries}{No\ of\ binaries\ with\ ngram\ in\ it}$$

5.2.4 Machine learning prediction engine

The output of the feature database is fed to the machine learning prediction engine. The dataset contains the feature vector values of each binary. Processing millions of assembly instructions takes polynomial time using the traditional programming approach, so I adopted an extensive data computing framework and used Apache Spark to train and test our labeled dataset.

5.2.5 Resampling

Resampling methods involve the repeated drawing of samples and reanalyzing the model. I selected K-fold cross-validation for resampling because of its broader acceptance rate by the research community. K-fold is the statistical method to compare and choose a model for a predictive modeling problem.

5.2.6 Supervised Classifiers

I applied various supervised machine learning algorithms to ransomware and benign labeled dataset. I use Naive Bayes, Logistic Regression, SVM, Random Forest, and Decision Tree, leveraging the Mlib spark library.

5.2.7 Model Fitting and Evaluation

I used supervised machine learning classifiers along with the training and test dataset. The model accepts only those classifiers which have accuracy greater or equal to the given threshold value.

5.3 Experiments and analysis

In this section, I discuss dataset collection, experimental protocol, and experimental results.

5.3.1 Dataset

The dataset for the experiment was collected from various sources, such as Virus Total and open-source malware repository theZoo [44]. I used 292 only ransomware binaries and the same number of benign executables for our experiment.

5.3.2 Experimental protocol

In this work, I use Apache Spark to do the big data processing of n-gram sequences. Apache Spark provides Mlib library [5] to implement various machine learning algorithms.

A brief description of Spark, cluster configuration, and feature table follows the below sections.

5.3.3 Apache Spark

Apache Spark is the popular distributed big data processing framework, which consists of Spark core and a set of libraries. It has libraries for SQL queries, streaming, machine learning, and graph processing. Spark core is responsible for managing our submitted job, i.e., it manages the handling of data, processing, execution, and result delivery.

Different components of the Apache Spark framework are shown in Figure 42.

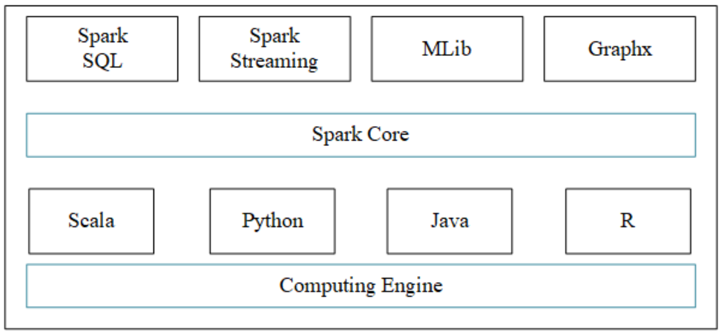


Figure 22: Basic architecture of Apache Spark framework

Spark processing is faster than Hadoop due to its Resilient Distributed Dataset (RDD), which supports in-memory processing computation. The state of memory is stored as an object across the jobs, and the object is shareable between those jobs [6]. Researchers mostly use Apache Spark to support the machine learning model and real-time processing for their malware analysis job. Figure 23 shows the basic building blocks for malware detection using the Apache Spark framework.

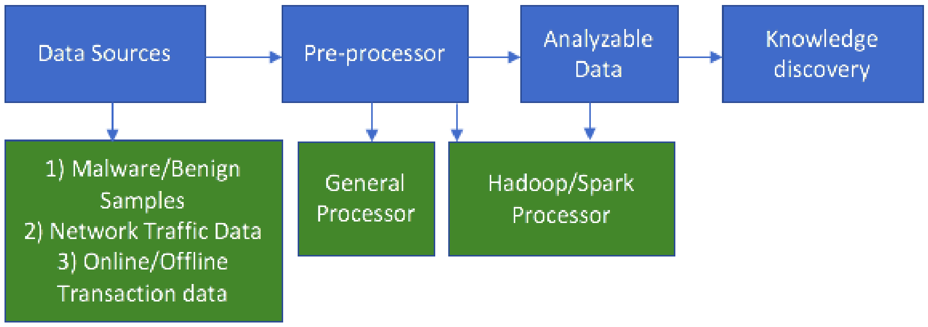


Figure 23: Malware analysis using Bigdata framework

Data sources can be malware/benign executable files, network traffic data, or online/offline transaction data from where we want to detect an anomaly or unusual pattern. The pre-processor component is the most powerful component of this architecture. The data sources go into pre-processing tasks using normal computing processing capabilities or are solely done using Hadoop/Spark frameworks. The result of this is clean and analyzable data, where we apply various machine learning or data mining algorithms to find the helpful pattern, also termed knowledge discovery.

5.3.4 Cluster configuration

I used the Apache spark cluster with the following configuration. There are 4 data nodes and one name node, each with 16GB RAM and eight cores, Ubuntu 16.04.3 operating system, and 1TB disk. Hadoop version-2.7.3 and Spark-2.3 are used.

5.3.5 Feature table

The Table 7 shows the distinct number of n-gram sequence features at different levels.

The total number of features differs for different N values of n-grams. As the value

Table 7: Number of unique N-gram features at multi-level

N-gram(N)	DLL	Function call	Assembly
2	2035	24,416	71,999
3	2797	29,226	1,539,769
4	2874	30,483	7,198,017
5	2842	30,962	12,038,570
6	2746	31,196	14,147,291

of N increases, the number of features also increases in the function call and assembly level. It is slightly irregular in the DLL level. From this table, we can claim that there is less overlapping of features as we increase the value of N.
All in all, I report the performance of the proposed framework in terms of accuracy. My prior published work [102] reported other performance matrices and results, including false positives, which are not compared here.

5.3.6 Experimental Results

I performed four experiments at different levels. I also analyzed the top ten trigrams based on their n-gram probability scores. More details are provided in the following sections.

5.3.6.1 Performance analysis of ML Malware detectors

The first three experiments shown in Tables 8, 9, and 10 is based on n-gram probability scores while the Table 11 is based on n-gram TF-IDF score.

Table 8: Experiment 1: Machine learning algorithms accuracy evaluation for n-gram probabilities at Dll level

Machine learning algorithm	N=2	N=3	N=4	N=5	N=6
Naive Bayes	82.19	75.34	73.97	73.28	72.43
Logistic Regression	89.55	88.43	85.44	84.93	82.70
SVM	88.52	86.98	85.1	84.58	82.87
Random Forest	86.64	85.27	85.27	84.76	83.4
Decision Tree	81.67	78.59	72.6	71.4	71.4

Table 9: Experiment 2: Machine learning algorithms accuracy evaluation for n-gram probabilities at Function level

Machine learning algorithm	N=2	N=3	N=4	N=5	N=6
Naive Bayes	85.62	80.39	79.73	79.08	79.08
Logistic Regression	93.25	92.06	91.28	92.81	91.50
SVM	92.16	81.52	69.02	60.86	57.06
Random Forest	91.50	91.06	89.97	85.94	82.02
Decision Tree	74.83	72.54	65.68	65.68	61.11

Table 10: Experiment 3: Machine learning algorithms accuracy evaluation for n-gram probabilities at Assembly level

Machine learning algorithm	N=2	N=3	N=4	N=5	N=6
Naive Bayes	76.77	74.13	72.44	70.98	70.01
Logistic Regression	78.43	80.24	79.11	77.5	76.8
SVM	76.91	76.95	75.49	75.2	73.19
Random Forest	80.1	80.056	79.88	79.7	78.46
Decision Tree	79.82	79.68	76.66	75.04	73.2

Table 11: Experiment 4: Logistic regression accuracy evaluation for n-gram TF-IDF at multi level

Level	N=2	N=3	N=4	N=5	N=6
Dll	93.36	81.52	69.02	60.86	57.06
Function call	96.08	98.04	90.20	84.31	72.55
Assembly Instruction	77.14	83.33	81.67	80	80

Table 12: Experiment 5: Logistic regression accuracy evaluation for n-gram TF-IDF at Combined multi level

Level	N=2	N=3	N=4	N=5	N=6
Dll, Function call and Assembly	97.13	98.59	90.45	85.11	72.58

At the DLL level, the highest accuracy is found to be 89.55% for Logistic regression at N=2. SVM has the second-best performance with 88.52% at N=2. The accuracy is found to be in a decreasing order while increasing the value of N. At the function call level, the highest accuracy is found to be 93.25% at N=2 for the logistic regression classifier. SVM follows with 92.16%. A similar trend is observed at the assembly level. Logistic regression with 80.24% accuracy at N=3 is the best-observed detection accuracy at this level.

Table 13: Top 10 Trigram sequences at different levels

Ransomware binaries					
DLL		Function call		Assembly Instruction	
Trigram	Score	Trigram	Score	Trigram	Score
ntdll, kernel32, comctl32	1.0	0CReaderWriterLock, 0CSingleList, 0CSmallSpinLock	1.0	sha256msg2, xor, or	0.33
msdart, mlang, midimap	0.5	InSendMessageEx, DialogBoxParamA, SetMenuItemBitmaps	1.0	addss, mov, mov	0.33
msdart, mlang, advapi32	0.5	TabbedTextOutW, ReleaseDC, GetDC	1.0	vpmacssww, push, daa	0.33
dsauth, gdi32, mstask	0.5	AddAccessDeniedAce, AreAnyAccessesGranted, GetCommandLineA	1.0	wrpkru, cld, mov	0.33
kernel32, user32, advapi32	0.40	__vbaVarSub, _CIcos, _adj_fptan	1.0	kmovd, pushf, lds	0.33
wtsapi32, psapi, msvcrt	0.33	DbgPrint, LdrGetProcedureAddress, RtlInitAnsiString	1.0	vpxorq, xchg, ror	0.33
winhttp, comctl32, shlwapi	0.33	BuildSecurityDescriptorW, RegSetValueW, RegConnectRegistryA	0.5	vfrczpd, in, and	0.33
msimg32, iphlpapi, oledlg	0.33	DuplicateToken, CreateServiceA, SetSecurityDescriptorOwner	0.5	mulss, xchg, aas	0.33
midimap, icmp, mfcsubs	0.33	LdrGetProcedureAddress, RtlInitAnsiString, LoadLibraryW	0.5	mwait, je, push	0.33
msacm32, kernel32, glu32	0.33	_lopen, LoadLibraryW, GetConsoleCP	0.375	vtestps, imul, add	0.33
Normal binaries					
DLL		Function call		Assembly Instruction	
Trigram	Score	Trigram	Score	Trigram	Score
api-ms-win-core-crt-l1-1-0, api-ms-win-core-crt-l2-1-0, api-ms-win-core-libraryloader-l1-2-0	1.0	SetupDiGetDeviceInstanceIdW, SetupDiDestroyDeviceInfoList, SetupDiEnumDeviceInfo	1.0	sgdtd, jne, push	0.33
dnssd, ws2_32, kernel32	1.0	SkciInitialize, SkciQueryInformation, SkciTransferVersionResource	1.0	vcmpltps, add, sub	0.33
iumcrypt, api-ms-win-core-heap-obsolete-l1-1-0, api-ms-win-eventing-cp-l1-1-0	0.5	UnregisterPowerSettingNotification, DispatchMessageW, MsgWaitForMultipleObjects	1.0	vpmacsdqh, enter, in	0.33
ntdsapi, logoncli, rpcrt4	0.5	SkciQueryInformation, SkciTransferVersionResource, SkciValidateDynamicCodePages	0.5	cmpxchg8b, retf, lock	0.33
esent, ntdll, api-ms-win-core-file-l1-1-0	0.5	ChooseFontW, GetSaveFileNameW, InitCommonCtrlEx	0.5	vpminuw, cwde, pop	0.33
tapi32, gdi32, user32	0.33	AddSIDToBoundaryDescriptor, CreateBoundaryDescript, CreatePrivateNamespaceW	0.33	vcvtsd2usi, dec, jge	0.33
mshtml, urlmon, msiso	0.33	DeleteBoundaryDescriptor, OpenPrivateNamespaceW, GetSecurityDescriptorDacl	0.33	vpshaw, ret, movabs	0.33
mswsock, ws2_32, winmm	0.33	EnterCriticalPolicySection, DeviceIoControl, GetSystemTimeAsFileTime	0.33	pinsrb, test, je	0.33
dpx, ntdll, ole32	0.33	LogonUserExW, WaitServiceState,EncodePointer	0.33	vpminsd, xor, rex	0.33
kerbclientshared, ntlmshared, msasn1	0.33	RtlAddAccessDeniedAce, NtOpenKey, NtQueryKey	0.33	vfnmsubpd, jrcxz, jge	0.33

Table 11 shows the performance evaluation for n-gram TF-IDF at multi-level using Logistic regression. Since the above three experiments showed the best performance for Logistic regression, I evaluated the TF-IDF feature set of n-grams using this classifier. The highest achieved accuracy rate is 93.36% at N=2 for DLL level, 98.04% at N=3 for function call level, and 83.33% at N=3 for Assembly level. The average accuracy for multi-level at N=2 is 88.86% and 87.63% at N=3. Table 12 shows the accuracy at combined multi-level using Logistic regression. The result shows the improved accuracy, which is a gain of combined multi-level analysis. The highest accuracy is achieved at N=3 with 98.59%, followed by 97.13% at N=2.

5.3.7 Analysis of top 10 Trigrams at different levels for ransomware and normal binaries

Table 13 shows the top ten DLL sequences for ransomware and normal executables along with their n-gram probability scores. The observed trigram sequence with a score of 1.0 signifies the surety of that particular sequence to be called in order. Trigram sequence ntdll, kernel32, comctl32 are expected to occur starting with ntdll. Kernel32, which is a part of that sequence, can be a starting sequence for other trigrams. There is a 40% probability that the sequence kernel32, user32, advapi32 will occur starting with kernel32. The top trigram sequences for ransomware and benign binaries differ significantly. For example, kernel32, user32, advapi32 has a score of 0.40 in ransomware samples, whereas the trigram sequence is different with a different score for benign samples. It is found to be the sequence: advapi32, kernel32, user32 with a score of 0.192 (Not shown in table). This typical behavior is seen with other trigrams as well.

The function call level n-grams occur with different n-gram sequences in ransomware samples. The sequence AddAccessDeniedAce, AreAnyAccessesGranted, GetCommandLineA with 1.0 for ransomware sample is sure to happen and is different than sequence RtlAddAccessDeniedAce, NtOpenKey, NtQueryKey with 0.33 in normal sample. We can observe the different sequence dependencies of each function calls in ransomware and normal binaries.

Similar to the n-gram patterns described in the above two levels, the n-gram sequences at the assembly level exhibit distinguishing behavior. Trigram pattern sha256msg2, xor, or is seen in ransomware with a score of 0.33, while a different pattern vpminsd, xor, rex with 0.33 is seen in normal binaries. Though we found the same instruction(xor) in both the samples, their sequences were different with the same or different scores. So, I claim that these distinct sequences built the unique feature set to achieve high detection rates using various machine learning classifiers.

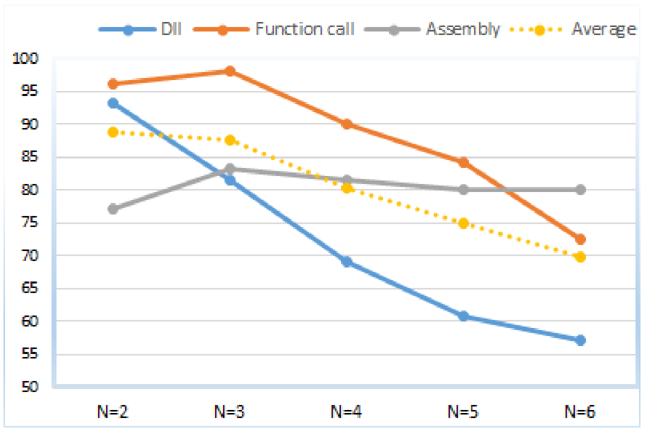


Figure 24: Logistic regression accuracy for N-gram TF-IDF at multi-level

The Figure 24 shows the accuracy graph for all three levels for n-gram TF-IDFs. I also calculated the average accuracy among these three levels. Among three levels, function call achieved improved high accuracy. The accuracy rate at N=3 is about 2% more than N=2, but there is a smooth decrease at other higher values of N. There is a steep decrease in accuracy for the Dll level. The detection rate of 93.36% at N=2 for DLL level decreases to 81.52% at N=3 and finally to 57.06% coming at N=6. The decrease is more rapid than the other two levels. Accuracy at the assembly level has a different pattern, it has improved accuracy at N=3, but the accuracy decreases slightly and becomes constant at N=5 and N=6. The average graph line shows that the accuracy is inversely proportional to N's value.

5.4 Summary

In this work, I proposed a multi-level ransomware detection framework in a big data platform leveraging the NLP domain, machine learning, and reverse engineering techniques. I experimented with ransomware at different code levels, flowing from DLL to function call and then to assembly instructions level to better understand various components and payloads. I used an Apache Spark computing environment for faster processing, but a general-purpose computer can also be used.

The multi-level analysis produces improved detection results compared to the individual levels. The highest detection accuracy for n-gram TF-IDF at N=3 is 98.59%, followed by 97.13% at N=2. I found that the empirical results of the multi-level analysis are convincing for further research to detect emerging ransomware effectively.

6 Hybrid Analysis Technique for Ransomware Detection

This work proposes an AI-based ransomware detection framework using a hybrid (combination of both static and dynamic) malware analysis techniques. Dynamic binary instrumentation is done using the PIN tool; function call trace is analyzed leveraging Cuckoo sandbox and Ghidra. Features extracted at DLL, function call, and assembly level are processed with NLP, association rule mining techniques, and fed to different machine learning classifiers. Support vector machine and Adaboost with J48 algorithms achieved the highest accuracy of 99.54% with 0.005 false-positive rates for a multi-level combined term frequency approach.

6.1 AI-powered ransomware detection Framework

The overall back end architecture for AI-powered ransomware detection framework is shown in Figure 27. A brief discussion of each phase follows below.

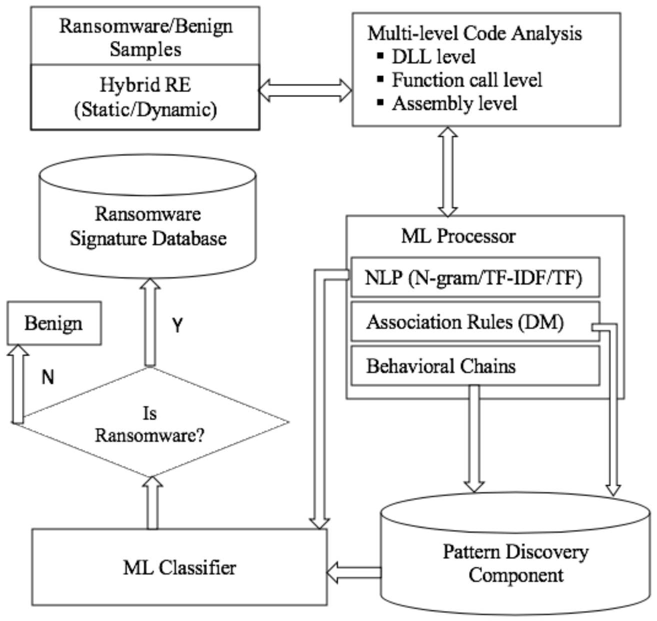


Figure 25: AI-powered ransomware detection framework. Some terms used are NLP: Natural Language Processing DM: Data Mining RE: Reverse Engineering.

6.1.1 Hybrid Reverse Engineering

First, the ransomware and benign samples will be reverse engineered using a hybrid approach. Hybrid reverse engineering involves both static and dynamic analysis of ransomware and benign binaries. This static and dynamic analysis comes with pros and cons. Static analysis is observing and extracting some features of a binary placed or stored in a hard drive. In contrast, dynamic analysis involves extracting behavior and features by running the binary in the memory. Static analysis is necessary to capture the initial properties of the binary. Further analysis is done using Object dump, a Linux-based tool, and PE parser [30], an open-source tool that helps to reveal properties like import and export of functions by the binaries. It is assumed to achieve high code coverage than via dynamic analysis because some missing parameters misguide the dynamic analysis. Static analysis generally tracks the code from start to end though the dynamic behavior is not captured. Dynamic analysis is done using a virtualized environment such as a Cuckoo sandbox [17] and the dynamic binary instrumentation tool, PIN [32]. The modern version of ransomware families is often difficult to analyze due to their anti-analysis techniques. Thus, I see the significance of hybrid analysis, and so is my approach. This phase also includes the initial pre-processing of the received raw outputs obtained via the adopted reverse engineering approaches.

6.1.1.1 Dynamic Binary Instrumentation

Programmers have used instrumentation techniques to diagnose program crashes, analyze errors, and write trace information. Instrumentation comes in two types: code instrumentation and binary instrumentation. Since the malware code is not available, I consider only binary instrumentation. Binary instrumentation is often referred to as Dynamic binary instrumentation or DBI in short, as the instrumentation is done using dynamic analysis of program traces. In our approach, I use the

PIN tool [32] for DBI. PIN makes tracking of every instruction executed possible by taking complete control over the run-time execution of the binary.

6.1.1.2 Cuckoo sandbox

Cuckoo sandbox is an advanced open-source automated malware analysis tool that allows execution and analysis of malware samples in a safe environment [17]. It makes use of the virtualization technique to run malware samples. I choose the Cuckoo sandbox because of its modular design supporting multiple environments and allowing flexibility in result analysis. If highly sophisticated ransomware does not run in a virtualized environment, then its function call trace is studied via reverse engineering provided by NSA's Ghidra tool [21].

6.1.2 Multi-level Code Analysis

Multi-level code analysis is a unique approach since the code is inspected at three levels i.e DLL, function call, and assembly level using a hybrid reverse engineering approach. I extract the binary behavior and properties specific to these three levels.

6.1.2.1 DLL Level

DLLs are dynamic link libraries that are subroutines to perform actions such as file system manipulation, navigation, process creation, communication, etc. They are loaded into the memory whenever required and freed from memory whenever not needed making our system lightweight. Thus, it makes effective use of available memory and resources by dynamic linking capability. DLLs have functions that they export and make available to other programs. During a program run, all necessary DLLs are loaded into the memory, but the referenced function call is accessed only when needed by locating the memory address where the function code resides. There are specific DLLs that are called more often because the function calls implemented by them are significant to carry out actions as per malware behavior.

6.1.2.2 Function call Level

A function call is a piece of code that has lines of instructions that impact the system or user. These are essential code blocks that carry out various functionalities and have less overlapping than DLL and assembly level analysis. Analysis at this level helps to identify function calls that are unique to malware's behavior. Categorizing functions based on functionality such as file operations, system information gathering, file enumeration, encryption key generation, encryption, etc., are some of the critical behaviors specific to ransomware that I analyze at this level.

6.1.2.3 Assembly Level

Assembly instruction is a low-level machine instruction, which is also called machine code. It can be directly executed by a computer's central processing unit (CPU). Each assembly instruction causes a CPU to perform a specific task, like *add, subtract, jump, xor*, and so on. Each function call or system call is implemented via assembly instructions. Assembly instructions are also analyzed based on categorized groupings. Some of the categories are Data transfer, Logical, Control transfer, Flag control, etc.

6.1.3 ML Processor

Machine Learning (ML) Processor consists of various machine learning components that assist the multi-level code analysis.

Natural Language Processing (NLP) language models have proved helpful in recommendation systems, text classification, speech recognition, and so on. Through the NLP component, various popular NLP techniques such as N-gram, Term Frequency-

Inverse Document Frequency (TF-IDF), and Term Frequency (TF) are leveraged to generate a feature database that is fed to the ML classifier.

Association rule is a rule-based data mining (DM) approach to discover notable relations and patterns among variables in a given data. FP-growth algorithm is preferred as it is more efficient than others, including Apriori. Apriori takes more execution time for repeated scanning to mine frequent items, but FP-growth scans the database only twice for constructing a frequent pattern tree. The FP-growth algorithm is leveraged to discover notable relations and patterns at a multi-level through the Association rules component.

Through the Behavior chain component, ransomware-specific chains are discovered, showing the relation at three levels.

6.1.4 Pattern Discovery Component

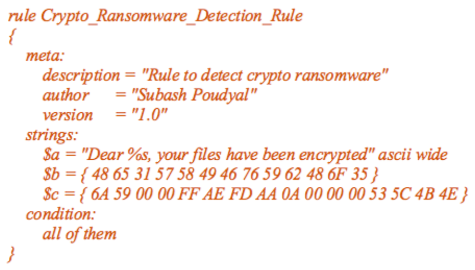
Both Association rules and Behavior chain component contribute to pattern discovery. The pattern database consists of all the discovered patterns, either association rule chains or behavioral chains. This database is considered as a feature database for the ML classifier.

6.1.5 ML Classifier

ML classifier component leverages the various supervised machine learning classifiers to train, test, and validate the model and decide whether a binary in consideration is ransomware or a benign application. I use Logistic Regression (LR), Support vector machines (SVM), Random Forest (RF), J48, and Adaboost with RF and J48.

6.1.6 Ransomware Signature Database

If the binary is ransomware, its signature will be stored in the malware signature database, and the binary is deleted. This decision will be based on a threshold accuracy of 95%. If the binary is not ransomware, then it is labeled as benign. The unique N-gram sequences with N-gram TF-IDF probability scores 1 are more certain to be seen. Similarly, association rules patterns with score value 1 are more certain to be seen. So, this is the case with custom behavioral chains. These patterns' corresponding opcode locations at the PE file help to set up conditions for the Yara rules [51]. Figure 26 shows a sample Yara rule. This rule says that if all the string's patterns specified in variables a, b, and c are observed in a binary, it is probable crypto-ransomware.



```
rule Crypto_Ransomware_Detection_Rule
{
   meta:
      description = "Rule to detect crypto ransomware"
      author      = "Subash Poudyal"
      version     = "1.0"
   strings:
      $a = "Dear %s, your files have been encrypted" ascii wide
      $b = { 48 65 31 57 58 49 46 76 59 62 48 6F 35 }
      $c = { 6A 59 00 00 FF AE FD AA 0A 00 00 00 53 5C 4B 4E }
   condition:
      all of them
}
```

Figure 26: Sample Yara rule to detect crypto ransomware

6.2 Dataset and experiments

The dataset consisted of the binaries in a portable executable (PE) file format. Most of the malware attacks occur by leveraging the PE file format. The bad actors often target the windows operating system (OS), which uses PE file format, and this OS is widely used all over the world. As per the security report from Fireeye [26], 70% of the malware attacks are launched via executable, which has PE file format. Five hundred fifty samples of ransomware were collected from Virus Total [45] and 540 normal samples from the Windows 10 OS and open-source software.

6.2.1 Experimental Protocols and Evaluation Measures

I experimented with coding in python, bash script, and the system with configuration Intel(R) Core(TM) i7-5500U CPU @ 2.40 GHz 2.39 GHz, 8.00 GB RAM, and 1 TB disk space. For running malware samples, six virtual environments were set up with five i7 processor machines, one with 32 GB RAM and four with 8 GB RAM.
I used widely used performance metrics of True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F-measure, and accuracy to evaluate the experiments performed. For association rule mining, the minimum support threshold is set to 2 and the confidence threshold to 0.8, but only association rules with a score of 1 are shown in section 1.6.

6.2.2 Experimental Results

The experiments done with combined multi-level features with term frequency show good accuracy with low false-positive rates as seen in Table 14. SVM and Adaboost with J48 reported the highest accuracy of 99.54% and lowest false positive rate of 0.005. J48 achieved the second-highest accuracy of 99.26% and a false positive rate of 0.007. The improved accuracy seen here than our previous approaches [100, 102] is due to the hybrid reverse engineering techniques used, which builds a unique feature set.

6.3 Summary

In this work, I proposed an AI-powered ransomware detection framework using the techniques of reverse engineering, hybrid analysis, and machine learning. Leveraging dynamic binary instrumentation tool PIN, Cuckoo sandbox environment, and

Table 14: Machine learning algorithms' evaluation for multi-level combined approach with term frequencies

Machine learning Classifier	TPR	FPR	Precision	Recall	F-measure	Accuracy (%)
Logistic Regression	0.992	0.008	0.992	0.992	0.992	99.17
Support Vector Machine	0.995	0.005	0.995	0.995	0.995	99.54
Random Forest(RF)	0.990	0.010	0.990	0.990	0.990	98.99
J48	0.993	0.007	0.993	0.993	0.993	99.26
Adaboost with RF	0.987	0.013	0.987	0.987	0.987	98.71
Adaboost with J48	0.995	0.005	0.995	0.995	0.995	99.54

Ghidra framework, I generated a distinguishing feature dataset and achieved high accuracy and low false-positive rate. Association rule mining and Ghidra's disassembly contributed to the existing analysis by other approaches to build unique behavioral multi-level chains specific to ransomware. Collectively this contributes to creating unique Yara rules which researchers and the security community can use.

7 Analysis of Crypto-Ransomware using ML-based Multi-level Behavior Profiling

In this work, I developed an AI-powered hybrid approach overcoming the recent challenges to detect ransomware. Specifically, I proposed a deep inspection approach for multi-level profiling of crypto-ransomware, which captures the distinct features at DLL (Dynamic-link library), function call, and assembly levels. I showed how the code segments correlate at these levels for studied samples. Our hybrid multi-level analysis approach includes advanced static and dynamic methods and a novel strategy of analyzing behavioral chains with AI techniques. Moreover, association rule mining, natural language processing techniques, and machine learning classifiers are integrated for building ransomware validation and detection model. I experimented with crypto-ransomware samples (collected from VirusTotal). One machine-learning algorithm achieved the highest accuracy of 99.72% and a false positive rate of 0.003 with two class datasets. The result exhibited that multi-level profiling can better detect ransomware samples with higher accuracy. The multi-level feature sequence can be extracted from most applications running in the different operating systems; therefore, I believe that our method can detect ransomware for devices on multiple platforms. I designed a prototype, AIRaD (AI-based Ransomware Detection) tool, which will allow researchers and the defenders to visualize the analysis with proper interpretation [99].

7.1 Hybrid Reverse Engineering at Multiple Levels

Advance reverse engineering techniques are applied at multiple levels, i.e., Dll, function call, and assembly. Details about the methods and multiple levels are discussed in the subsections given below.

7.1.1 Hybrid Reverse Engineering

Reverse engineering (RE) is a backward process of re-creating things. It deals with disassembling and dealing with each piece of disassembled components to find the behavior of the original one. In malware analysis, the actual code is rarely known, and there comes the need for RE to see the actual conduct of the malware to detect and defend against its attack.

Ransomware analysis is done either using static, dynamic, or hybrid approaches. Static analysis is the basic one that involves inspection of the binary stored in a hard disk or drive. This analysis reveals compile-time, binary format, imports, exports, strings used, etc. However, the run-time behavior of the ransomware cannot be captured. This shortcoming is achieved using dynamic analysis where a malware sample is run in the memory, and its behavior is revealed to the analyst. A hybrid analysis is the combination of both of these approaches [109, 118].

First, the ransomware and benign samples are reverse engineered using a hybrid approach. Ransomware writers often use various notoriously clever techniques to bypass analysis techniques to evade the defenders. Though, some virtualization tools and environments claim they can overcome the anti-analysis techniques used by cybercriminals. But, this is not as easy as claimed. This is also one of the reasons why I adopted a hybrid approach. Another reason is the ability to capture distinct features and a reasonable detection rate which we will see in the coming sections of this work. Those binaries which do not run or exit in the middle will be analyzed using a PE parser [30], an open-source tool that helps to reveal properties like DLL used, import, and export of functions by the binaries. Dynamic analysis is done using a virtualized environment with Cuckoo sandbox [17], advanced disassembler, and debugger tool from National Security Agency Research Directorate [21], and the dynamic binary instrumentation tool, PIN [32].

Below is a description of the tools and techniques I used for hybrid reverse engineering.

7.1.1.1 Dynamic Binary Instrumentation

Programmers have used instrumentation techniques to diagnose program crashes, analyze errors, and write trace information. Instrumentation comes in two types: code instrumentation and binary instrumentation. Since the malware code is not available, I consider only binary instrumentation. Binary instrumentation is often referred to as Dynamic binary instrumentation or DBI, in short, as the instrumentation is done using dynamic analysis of program traces. In this approach, I use the PIN tool [32] for DBI. PIN makes tracking of every instruction executed by taking complete control over the run-time execution of the binary.

7.1.1.2 Cuckoo sandbox

Cuckoo sandbox is an advanced open-source automated malware analysis tool that allows execution and analysis of malware samples in a safe environment [17]. It makes use of the virtualization technique to run malware samples. If highly sophisticated ransomware does not run in a virtualized environment, then its function call trace is studied via reverse engineering provided by NSA's Ghidra tool [21].

7.1.1.3 Ghidra

Ghidra is a reverse engineering framework developed by the National Security Agency Research Directorate [21]. This tool allows malware researchers to analyze binaries on a variety of platforms. It will enable various features, including disassembly and decompilation. Ghidra's disassembly features help to analyze the behavior of ransomware samples better. Its use made defining the multi-level chains of ransomware possible.

7.1.2 Feature Extraction at multiple levels

The raw output obtained from the hybrid reverse engineering techniques is pre-processed and fed to the feature extraction component. Features are extracted at three levels: DLL, function call, and assembly. Below is the description of the extractor at each level.

7.1.2.1 DLL level

DLL extractor at the DLL level works via an automated script that parses through the raw reverse engineered output. It gives the list of all DLLs being called by various function calls. DLLs are dynamic link libraries that are subroutines to perform actions such as file system manipulation, navigation, process creation, communication, etc. They are loaded into the memory whenever required and freed from memory whenever our system is lightweight. Thus, it makes effective use of available memory and resources by dynamic linking capability. DLLs have functions that they export and make available to other programs. During a program run, all necessary DLLs are loaded into the memory. Still, the referenced function call is accessed only when needed by locating the memory address where the function code resides. Specific DLLs are called more often because the function calls implemented by them are significant to carry out actions as per malware behavior.

7.1.2.2 Function call level

Function call extractor at function call level is implemented through an automated script that parses through the raw reverse engineered output. It lists all function calls being used by the program in execution, as seen in the execution trace. A function call is a piece of code that has lines of instructions that impact the system or user. These are essential code blocks that carry out various functionalities and have less overlapping than DLL and assembly level analysis. Analysis at this level helps to identify function calls that are unique to malware's behavior. Categorizing functions based on functionality such as file operations, system information gathering, file enumeration, encryption key generation, encryption, etc., are critical behaviors specific to ransomware that we analyze at this level.

7.1.2.3 Assembly Level

Dynamic binary instrumentation tool PIN is leveraged to get the assembly instructions being used by the program. Assembly instruction is a low-level machine instruction, which is also called machine code. It can be directly executed by a computer's central processing unit (CPU). Each assembly instruction causes a CPU to perform a specific task, like *add, subtract, jump, xor*, and so on. Each function call or system call is implemented via assembly instructions. Assembly instructions are also analyzed based on categorized groupings. Some of the categories are Data transfer, Logical, Control transfer, Flag control, etc.

7.2 Use of Machine learning

Traditional malware detection techniques are primarily based on signatures that adversaries can easily evade. Machine learning systems have better detection capability as they can automate the work of creating signatures. Moreover, they can better detect the previously unseen malware.

7.2.1 Data mining techniques

Data mining is the process of finding hidden features or patterns that can distinguish a given family sample from others. Various approaches are leveraged for data mining.

7.2.1.1 Frequency analysis

Frequency analysis is a simple but powerful data analysis technique. It gives various insights about the most used or least used component.

7.2.1.2 Association rules

Association rules show the probability of the relationship between items. It is generally a rule-based data mining approach that helps to discover correlations among data items. In this approach, I have used the FP-Growth algorithm, which is array-based, uses depth-first search, and requires only two database scans, making it more efficient and scalable. A list of DLLs, function calls, and corresponding assembly instructions obtained from the advanced reverse engineering process is provided as an input to the FP-Growth algorithm. The generated FP-Growth association rules are used to create ransomware detection signatures.

Table 15: Association rule mining at DLL level with score: 1.0

Association rules at DLL level
[ADVAPI32, OLE32, SHELL32, WS2_32] → [KERNEL32]
[ADVAPI32, OLE32, SHELL32, WS2_32] → [ADVAPI32]
[ADVAPI32, OLE32, SHELL32, WININET] → [KERNEL32]
[ADVAPI32, OLE32, SHELL32, WININET, WS2_32] → [KERNEL32]
[KERNEL32, OLE32, SHELL32, WININET, WS2_32] → [ADVAPI32]
[ADVAPI32, KERNEL32, OLE32, SHELL32, WS2_32] → [WININET]
[ADVAPI32, KERNEL32, OLE32, WININET, WS2_32] → [SHELL32]
[ADVAPI32, KERNEL32, OLE32, SHLWAPI, WININET, WS2_32] → [SHELL32]
[ADVAPI32, KERNEL32, OLE32, SHELL32, SHLWAPI, WININET] → [WS2_32]
[ADVAPI32, KERNEL32, OLE32, SHELL32, WININET, WS2_32] → [SHLWAPI]
[ADVAPI32, CRTDLL, GDI32, KERNEL32, OLE32, OLEAUT32, USER32] → [WININET]
[ADVAPI32, CRTDLL, GDI32, KERNEL32, OLE32, OLEAUT32, WININET] → [USER32]
[ADVAPI32, CRTDLL, GDI32, OLE32, OLEAUT32, USER32, WININET] → [KERNEL32]

7.2.2 Association rules at DLL level

Table 15 shows a portion of the association rules at the DLL level. These DLLs are obtained from the import table of ransomware portable executable files. The first-row rule shows that ADVAPI32, OLE32, SHELL32, and WS2_32 DLL imply KERNEL32 DLL. This rule contains DLLs associated with ransomware-specific behavior, including encryption as described in section 7.3.1. The eighth rule shows that ADVAPI32, KERNEL32, OLE32, SHLWAPI, WININET, WS2_32 implies SHELL32. This rule contains DLLs specific to ransomware behavior, including the deletion of a shadow copy.

Table 16: Association rule mining at function call level with score: 1.0

Association rules at function level
[GetCurrentThreadId, GetTickCount, RtlUnwind, WideCharToMultiByte, lstrlenW] → [ExitProcess]
[CloseHandle, GetModuleHandleA, GetTickCount, ReadFile, RtlUnwind, WideCharToMultiByte] → [GetProcAddress]
[CloseHandle, GetCurrentProcessId, GetModuleHandleA, ReadFile, RtlUnwind, WideCharToMultiByte] → [ExitProcess]
[ExitProcess, GetCurrentThreadId, GetTickCount, SetFilePointer, Sleep, WideCharToMultiByte] → [GetModuleHandleA]
[ExitProcess, GetCurrentThreadId, GetModuleHandleA, RtlUnwind, SetFilePointer, WideCharToMultiByte] → [WriteFile])
[CloseHandle, ExitProcess, GetCurrentProcessId, GetModuleHandleA, RtlUnwind, WideCharToMultiByte] → [ReadFile]
[GetCurrentProcess, InterlockedIncrement, IsDebuggerPresent, RaiseException, RtlUnwind, SetUnhandledExceptionFilter, Sleep, UnhandledExceptionFilter, VirtualProtect, WideCharToMultiByte] → [HeapCreate]
[EnterCriticalSection, ExitProcess, GetLastError, GetProcAddress, LeaveCriticalSection, WriteFile] → [HeapReAlloc]
[DeleteCriticalSection, GetCurrentProcessId, GetLastError, GetTickCount, SetLastError, TlsAlloc, TlsFree] → [TlsSetValue]
[GetCurrentProcessId, GetStringTypeW, GetTickCount, VirtualProtect] → [InitializeCriticalSectionAndSpinCount]
[GetCurrentProcess, GetCurrentProcessId, GetTickCount, InitializeCriticalSectionAndSpinCount, IsDebuggerPresent, RaiseException, SetUnhandledExceptionFilter, UnhandledExceptionFilter, VirtualProtect] → [SetHandleCount]

7.2.3 Association rules at Function call level

Table 16 shows a portion of the association rules at the function call level. These function calls are obtained via both static and dynamic analysis of ransomware executables. The first row rule shows the use of GetCurrentThreadId, GetTickCount, RtlUnwind, WideCharToMultiByte, lstrlenW implies ExitProcess. This rule contains function calls that are specific to the anti-analysis behavior of ransomware.

Table 17: Association rule mining at assembly level with score: 1.0

Association rules at Assembly level
[add, and, cmp, data16, jmp, lea, sbb, shl] $\rightarrow$ [xor]
[add, cmp, data16, jm, lea, sbb, shl, sub] $\rightarrow$ [xor]
[add, cmp, data16, jmp, or, rcr, sbb] $\rightarrow$ [ror]
[add, cmp, data16, jmp, or, rcr, shl] $\rightarrow$ [sbb]
[add, cmp, data16, jmp, rcr, sbb, shl] $\rightarrow$ [xor]
[add, cmp, fdivr, jmp, les, or, repz, sbb, sub] $\rightarrow$ [bound, ror]
[add, cmp, fdivr, jmp, les, or, repz, shl, sub] $\rightarrow$ [sbb]
[add, cmp, fdivr, jmp, les, repz, sbb, shl, sub] $\rightarrow$ [xor]
[add, call, ficom, les, lock, or, rcr, sbb, xchg] $\rightarrow$ [cmp]
[call, cmp, ficom, les, lock, or, rcr, sbb, xchg] $\rightarrow$ [add]
[add, call, cmp, ficom, les, or, rcr, sbb, xchg] $\rightarrow$ [lock]
[add, call, cmp, ficom, lock, or, rcr, sbb, xchg] $\rightarrow$ [mul]
[add, call, ficom, les, lock, or, rcr, sbb, xchg] $\rightarrow$ [cmp]

7.2.4 Association rules at Assembly level

Table 17 shows a portion of the association rules at the assembly level. These function calls are obtained via dynamic binary instrumentation. These rules are specific to different function calls defined for various chains as shown in section 7.3.1.

The combination of association rules at three levels proves more effective to create unique ransomware detection signatures based on functionality chains discussed in section 7.3.1. The experimental results also show promising accuracy with low false-positive rates.

7.2.5 NLP techniques

Machine Learning (ML) processor makes use of natural language processing (NLP) language models. NLP has proved useful in recommendation systems, text classification, speech recognition, and so on. Through the NLP component, various popular NLP techniques such as N-gram, Term Frequency-Inverse Document Frequency (TF-IDF), and Term Frequency (TF) are leveraged to generate a feature database that is fed to the ML classifier.

7.2.6 ML Classifier

ML classifier component leverages the various supervised machine learning classifiers to train, test, and validate the model and decide whether a binary in consideration is ransomware or a benign application. The threshold accuracy is set to 95%. If the binary is classified as ransomware, then I generate an alert and delete the sample. Whereas if the binary is classified as not ransomware, then it is labeled as a benign application. I use Logistic Regression (LR), Support vector machines (SVM), Random Forest (RF), J48, Adaboost with RF/J48, and Neural network.

7.3 HMLP based detection and Behavioral Chaining

The computational blocks and flow diagram for hybrid multi-level profiling (HMLP) for ransomware detection are shown in Figure 27. Both ransomware and benign binaries are analyzed using hybrid reverse engineering techniques, and feature extraction at multi-level is done as discussed in section 7.1. The use of machine learning is discussed in section 7.2. Behavioral chaining is the core component of this framework and is discussed in detail below.

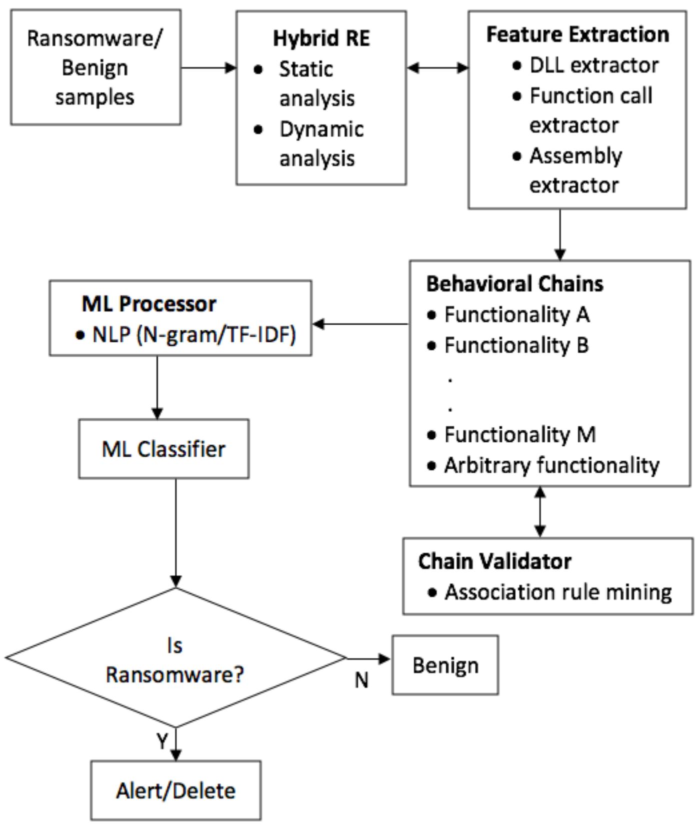


Figure 27: HMLP based ransomware detection framework

7.3.1 Behavioral Chaining

A chain is a continuous sequence of components that achieve a specific functionality or activity. Behavioral chains are collections of functionality chains at different levels or can be linked across a multi-level. Functionalities are derived from different ransomware families, which are analyzed. The algorithmic steps to find a behavioral chain are illustrated in algorithm 1.

According to the algorithmic steps, for each ransomware family, I performed reverse engineering to extract features to build behavioral chain profiles, also dynamic analysis is done running a binary in an isolated virtual environment. Dynamic analysis is real execution of a binary to study the actual behavior and functionality. If some binaries do not run in a virtualized environment, I study the behavior using static

Algorithm 1: HMLP behavioral chaining

```
procedure HMLP_BEHAVIORAL_CHAINING(PE binaries)
    Step 1: Manual_chaining_crypto_ransomware
    for one sample from each ransomware families in [Locky, TeslaCrypt, GandCrab, CryptoWall, Cerber, Petya] do
        Hybrid RE of binaries
        Extract features at multi-level
        Inspect/analyze run time execution trace
        Identify and define major functionality chains
        Add multi-level components to major chains if no minor chains are found
        Identify and define minor functionality chains for each major chains
        Add multi-level components to each minor sub chains
        For all remaining chains keep as arbitrary chains
    end for
    Step 2: Automatic_chaining_all_samples
    for each sample in dataset do
        Hybrid RE of binaries
        Extract features at multi-level
        FP-growth rule mining at multi-level
        For each chain/rule with confidence 0.8 and score 1
        Check if a rule matches with defined functionality chain from Step 1
        Add chain as a distinct chain to feature dataset if matched
        Add chain as an arbitrary chain to feature dataset if not matched
    end for
    return behavioral chains
end procedure
```

analysis techniques.

We can predict the functionality through disassembly and run trace information; moreover, a deep manual inspection of these results helps create functionality chains at DLL, function call, and assembly level. A self-deletion functionality of malware has multi-level chains as shown in table 18. A major chain can have minor sub-chains. For example, a self-delete chain contains shadow copy delete as sub-chains under the major chain. I ignore the functionality which is commonly seen in benign applications such as error handling chains. For all other unidentified functionality, I keep it as an arbitrary chain. The next stage is automatic chaining for all given sample binaries. I performed reverse engineering of each sample using our HMLP algorithm to extract the raw features at three levels.

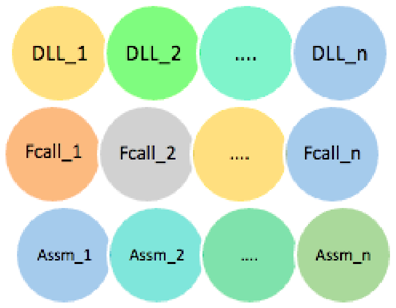


Figure 28: A sample arbitrary structure of a behavioral chain

I then leverage the FP-growth association rule mining algorithm. The association rules with support 0.8 and confidence score one are checked if found from the first stage. If found, they will be part of the final feature dataset. All non-matching chains with previously mentioned association rules scores become part of arbitrary chains in the final feature dataset for the given sample. An illustration of a behavioral chain is shown in Figure 28. Here, the acronym Fcall refers to function calls, and Assm refers to assembly instruction.

7.3.2 Relationship between Behavior chain and Association Rules

Association rules discover patterns in a given feature set, while behavioral chains are classification of those. I defined behavioral chains and sub-chains, which can be considered a semi-expert system. While many association rules are generated with varying lengths, properly labeling them in the behavioral chain is very important. In particular, behavior chains provide the explainability of the association rules.

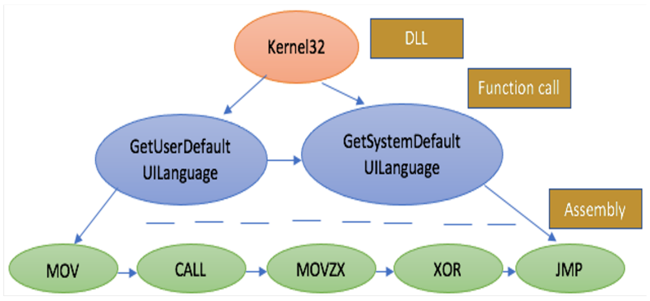


Figure 29: Behavioral chain for system profiling with default user and system language

Figure 29 shows the behavioral chain for system profiling, particularly for the identification of default user and system language, which is obtained via the relationship with the association rules. This 3-layered hierarchy is concerned with profiling system identifiers. Some of the other often profiled system identifiers are keyboard layout, windows version used, the domain used, CPU identifier, etc.

Figure 30 shows the behavioral chain for delete identification. If a major chain has variations due to implementation differences or minor functionality differences, then the major chain is broken down into all specifiable minor chains. Each minor chain will have some variation in either one or more levels of code analysis. The chain-building process is assisted by using association rule mining.

Based on the behavioral chain described in Figure 30 we can generate a ransomware signature as shown in Figure 31. This rule is based on Yara rule format [51]. Variable "a" contains the DLL components, "b" contains the function call components

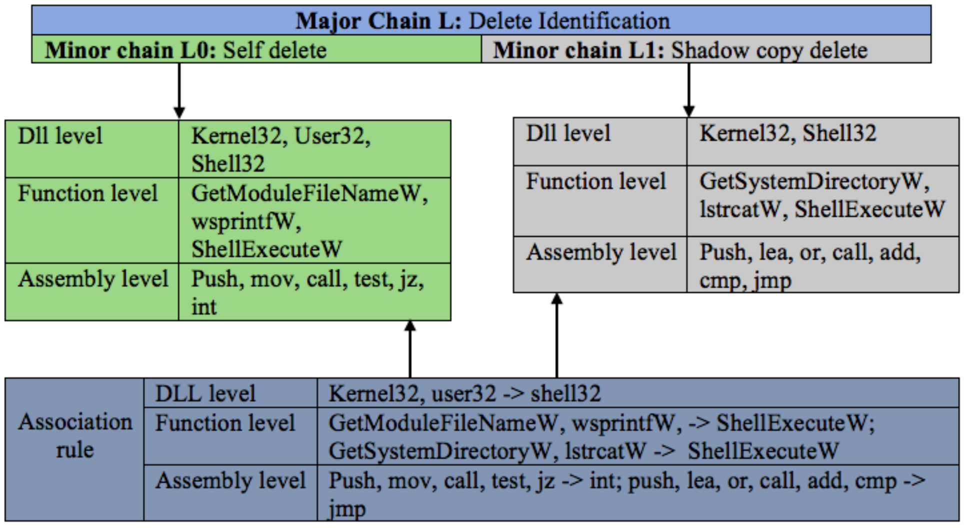
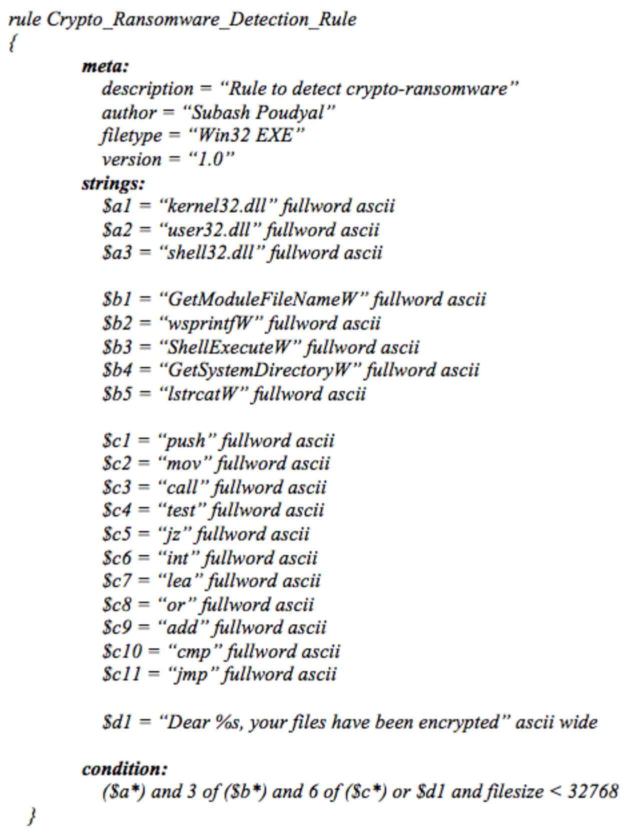

Major Chain L: Delete Identification
Minor chain L0: Self delete
Minor chain L1: Shadow copy delete
Dll level
Kernel32, User32, Shell32
Function level
GetModuleFileNameW, wsprintfW, ShellExecuteW
Assembly level
Push, mov, call, test, jz, int
Dll level
Kernel32, Shell32
Function level
GetSystemDirectoryW, lstrcatW, ShellExecuteW
Assembly level
Push, lea, or, call, add, cmp, jmp
Association rule
DLL level
Kernel32, user32 -> shell32
Function level
GetModuleFileNameW, wsprintfW, -> ShellExecuteW; GetSystemDirectoryW, lstrcatW -> ShellExecuteW
Assembly level
Push, mov, call, test, jz -> int; push, lea, or, call, add, cmp -> jmp

Figure 30: Behavioral chain for delete identification

and "c" contains the assembly components. The condition at last specifies that we need all from “a”, 3 among “b”, and 6 among “c” or “d” and filesize.

7.3.3 Chain Validator

Chain validator component aids in the automatic validation of the behavioral chains. I use association rule mining for this. Association rule is a rule-based data mining (DM) approach to discover notable relations and patterns among variables in a given data. FP-growth algorithm is preferred as it is more efficient than others, including Apriori. Apriori takes more execution time for repeated scanning to mine frequent items, but FP-growth scans the database only twice for constructing frequent pattern trees. I consider association rules with minimum support threshold two and confidence threshold of 0.8 and check whether it matches the defined chain ingredients. Only the matching chains form the functionality chains A-M. All non-matching chains with previously described support and confidence scores are part of arbitrary functionality chains. I use a novel approach to calculate the ransomware profiling chain ratio. Below, DCR represents the DLL chain ratio, FCR represents



```
rule Crypto_Ransomware_Detection_Rule
{
        meta:
          description = "Rule to detect crypto-ransomware"
          author = "Subash Poudyal"
          filetype = "Win32 EXE"
          version = "1.0"
        strings:
          $a1 = "kernel32.dll" fullword ascii
          $a2 = "user32.dll" fullword ascii
          $a3 = "shell32.dll" fullword ascii

          $b1 = "GetModuleFileNameW" fullword ascii
          $b2 = "wsprintfW" fullword ascii
          $b3 = "ShellExecuteW" fullword ascii
          $b4 = "GetSystemDirectoryW" fullword ascii
          $b5 = "lstrcatW" fullword ascii

          $c1 = "push" fullword ascii
          $c2 = "mov" fullword ascii
          $c3 = "call" fullword ascii
          $c4 = "test" fullword ascii
          $c5 = "jz" fullword ascii
          $c6 = "int" fullword ascii
          $c7 = "lea" fullword ascii
          $c8 = "or" fullword ascii
          $c9 = "add" fullword ascii
          $c10 = "cmp" fullword ascii
          $c11 = "jmp" fullword ascii

          $d1 = "Dear %s, your files have been encrypted" ascii wide

        condition:
          ($a*) and 3 of ($b*) and 6 of ($c*) or $d1 and filesize < 32768
 }
```

Figure 31: Ransomware signature based on delete identification behavior

a function call chain ratio, ACR represents the assembly chain ratio, and RPCR represents the ransomware profiling chain ratio.

$$DCR = \frac{No.\ of\ DLL\ chains\ seen}{Total\ no.\ of\ DLL\ chains}$$

$$FCR = \frac{No.\ of\ function\ call\ chains\ seen}{Total\ no.\ of\ function\ call\ chains}$$

$$ACR = \frac{No.\ of\ assembly\ chains\ seen}{Total\ no.\ of\ assembly\ chains}$$

$$MPCR = \frac{DCR\ +\ FCR\ +\ ACR}{Total\ number\ of\ levels}$$

Our HMLP detection approach chains are created at DLL, function call, and assembly level for given ransomware functionality. These chains are made leveraging the feature extraction component that relies on hybrid RE. I first manually inspect thousands of lines of function and activity trace of prominent ransomware families and reach a consensus to define the major chains seen in most ransomware families. For all other remaining samples, these chains are discovered and validated automatically using the chain validator component, which uses association rule mining.

Each main functionality chain can have sub-chains under them. If there is some overlapping in the main chain, it is kept under the sub-chain with some variations. If functionality cannot be well defined, then it is kept under an arbitrary functionality chain. I have identified chains from A to M, which incorporate most crypto-ransomware families based on crypto-ransomware behavior.

The multi-level analysis is crucial for creating unique ransomware signatures.

A brief explanation of each chain follows below.

Chain A deals with the initial setup.

Chain A0 uses *GetStartupInfoW* which gets information related to window station, desktop and appearance of the main window. *HeapSetInformation* enables features to use heap as a data structure. *HeapCreate* and *GetProcessHeap* are used by the HeapHandle parameter. *GetModuleHandleW* gets a handle for a given module, either an executable or a DLL file. This handle is referenced later to request the necessary function calls. *GetProcAddress* gets the memory address of an exported DLL function. The first parameter is the handle to the DLL and the second parameter is the function name exported from that DLL. For example, *CryptGenRandom* is exported from Advapi32 DLL. *FlsAlloc* allocates a fiber local storage index. Any fiber in the process can subsequently use this index to store and retrieve values that are local to the fiber [19]. It setups with three sub-functions which are *FlsGetValue, FlsSetValue* and *FlsFree.*

Chain A1 deals with console-setup. *GetStdHandle* gets a handle to the specified IO device. These handles are used by Windows applications to read and write to the console. These are also used by *ReadFile* and *WriteFile* functions. *GetEnvironmentStringsW* makes the environment variables available for the current process running in an infected computer. *FreeEnvironmentStringsW* frees all environment settings. This function is generally used only once. Malware writers do not want to interfere with their work. They may use *SetEnvironmentVariable* to set certain variables to fulfill their malicious behavior. *IsProcessorFeaturePresent* determines whether the system in use supports the specified processor feature.

Chain A2 deals with system services with error handling. *GetLastError* gets the last error code for the calling thread of the given process. Threads do not overwrite each other's error codes. *GetCurrentThreadId* gets the identifier value for the thread whose error code for execution of a particular function is to be considered. *SetLastError* sets the error code for the calling thread of a given process. For example, zero error code means error success, and the operation was completed. This sequence comes more often to get the status of functions being executed.

Chain A3 deals with module enumeration. *GetModuleFileName* loads the malware executable and *GetModuleHandle* gets the handle to the custom malware DLL with obfuscated functions. The obfuscation behavior can be captured at the assembly level.

Chain B deals with time tracking and anti-analysis behavior of the ransomware. *GetTickCount* gives the time in milliseconds that have passed since the system was started.

```
 mov ebx, 0xdbba0
call dword ptr [kernel32.dll::GetTickCount]
cmp eax, ebx
```

The 0xdbba0 value i.e 900000 milliseconds or 15 minutes is compared with the time obtained via *GetTickCount*. If the user system is active for less than 15 minutes, the malware will not execute, giving us a false impression of a benign application. Malware writers try to evade malware analysis being done in a virtual environment. Generally, such an analysis environment runs for less than 15 minutes or even less than that.

GetSystemInfo and *GetNativeSystemInfo* use *dwNumberOfProcessors* method to check the number of processors running in a system. If the system has only one processor then the malware writers label it as a analysis environment and may not execute at all. *GetSystemInfo* often invokes native low level function call *Nt-QuerySystemInformation* to get the number of cores used in a system.

Chain C deals with DLL and function loading. *LoadLibraryW* loads the specified DLL into the address space of the calling process. *GetModuleHandleW* gets the handle object for that particular DLL. *GetProcAddress* receives the address of an exported function from that specified DLL. This chain of function calls is the action triggering point of the code section while executing a binary.

Chain D deals with access elevation.

In **Chain D0**, *OpenProcessToken* opens the token associated with a given process while *GetTokenInformation* is used to obtain the token id, session id or security identifier of the process's owner. This obtained token is duplicated and applied to a new thread created in suspended mode using *SetThreadToken*.

Chain D1 is used to bypass user access control by elevating privilege to an admin level.

Chain D2 contains functions that destroy previously created handles for access and privilege escalation.

Chain E is used by ransomware for process enumeration. *CreateToolhelp32Snapshot* is used to create a snapshot of processes, heaps, threads, and modules. Malware often uses this function as part of code that iterates through processes or threads.

This snapshot function is called during different functionality blocks such as loading DLLs, loading application processes, loading anti-virus processes, etc. *Process32FirstW* gets information about the first process seen in a system snapshot. This is used to enumerate processes from a previous call to *CreateToolhelp32Snapshot*. Malware often enumerates through processes to find a process to inject into. Ransomware does not want applications running as it might affect their encryption operation as applications often lock the open or used files. *lstrcmpiW* compares all observed processes with the hard-listed ones to kill out; a similar comparison is made for observed anti-virus software to kill them. Some of the killed-out processes are mydesktopqos.exe, sqlbrowser.exe, sqlservr.exe, msftesql.exe, mysqld.exe, excel.exe, msaccess.exe, outlook.exe, winword.exe, wordpad.exe, etc. The malware process is often run under explorer.exe.

Chain F deals with parameter setup. The command-line string via *GetCommandLineA* serves as one parameter value to be passed to *GetCommandLineW* function which later removes malware itself and deletes shadow copies via the command prompt window.

Chain G deals with system profiling.

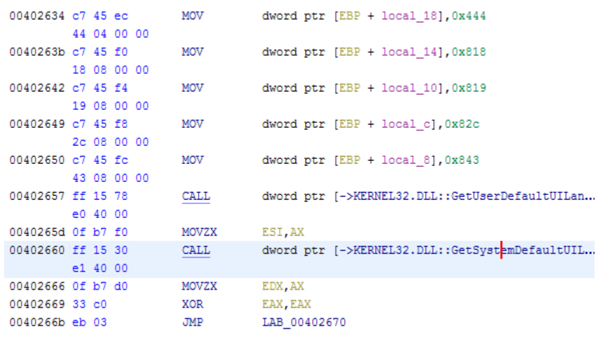
Chain G0 is concerned with profiling system identifiers. Some of the often profiled system identifiers are keyboard layout, windows version used, the domain used, CPU identifier, etc. *RegOpenKeyExW* opens the specified registry key for system profiling. The parameter *lpSubKey* specifies the name of the registry key to be open. The access right for registry key object is *KEY_EXECUTE* (0x20019) which is equivalent to *KEY_READ* and Combines the *STANDARD_RIGHTS_READ*, *KEY_QUERY_VALUE*, *KEY_ENUMERATE_SUB_KEYS*, and *KEY_NOTIFY* values.

When *lpSubKey*="Keyboard Layout Preload" it is inferred that malware is trying to know about keyboard layout. Similarly, other registry key values that are revealed include:

- *lpSubKey*="Control Panel International"
- *lpSubKey*="Keyboard Layout Preload"
- *lpSubKey*="SOFTWARE Microsoft Windows NT CurrentVersion"

RegQueryValueExW receives the type and data for the specified open registry key while the *RegCloseKey* closes all open handles of the registry keys.

Chain G1 deals with user interface language reveal. *GetUserDefaultUILanguage* gets the language identifier for the current user while *GetSystemDefaultUILanguage* gets for the operating system. This function chain is often used to reveal the user language so that the malware writers can decide whether to execute further or not based on the country and spoken language preferences. Figure 32 shows its usage.



```
00402634 c7 45 ec       MOV       dword ptr [EBP + local_18],0x444
         44 04 00 00
0040263b c7 45 f0       MOV       dword ptr [EBP + local_14],0x818
         18 08 00 00
00402642 c7 45 f4       MOV       dword ptr [EBP + local_10],0x819
         19 08 00 00
00402649 c7 45 f8       MOV       dword ptr [EBP + local_c],0x82c
         2c 08 00 00
00402650 c7 45 fc       MOV       dword ptr [EBP + local_8],0x843
         43 08 00 00
00402657 ff 15 78       CALL      dword ptr [->KERNEL32.DLL::GetUserDefaultUILan…
         e0 40 00
0040265d 0f b7 f0       MOVZX     ESI,AX
00402660 ff 15 30       CALL      dword ptr [->KERNEL32.DLL::GetSystemDefaultUIL…
         e1 40 00
00402666 0f b7 d0       MOVZX     EDX,AX
00402669 33 c0          XOR       EAX,EAX
0040266b eb 03          JMP       LAB_00402670
```

Figure 32: Disassembled code to check default user language

Chain G2 deals with checking malware footprint if it is already there in an infected system. If this footprint is already available, the malware will not execute; else, a new footprint is created. Generally, a locked file is created with a hidden attribute at C drive as a footprint. The 8 characters hex as a footprint is obtained from the volume serial number via *GetVolumeInformationW* function. A similar hidden footprint is left at each folder location before encryption. The *wsprintfW* writes the formatted data to the buffer while *CreateFileW* function creates a lock file as a footprint.

Chain G3 deals with creating a unique user id to identify the victim's system. A unique ransom id is calculated using the *RtlComputeCrc32* function, which uses the CPU name and the windows volume serial number as parameters. *RtlComputeCrc32* calculates the CRC32 checksum of a block of bytes. There may be other ways to create a unique user id, for example, based on a mac address or a combination of system information and a unique random number. Various cryptographic algorithms can be used as well.

Chain G4 deals with revealing information about disk usage. *GetDriveTypeW* determines whether a disk drive is a removable, fixed, CD-ROM, RAM disk, or network drive. Parameter *lpRootPathName* gives the root directory for the drive. This function returns a value from 0 to 6, 3 being fixed hard drive. *GetDiskFreeSpaceW* receives information about the given disk. It also reveals how much free space is available on the disk.

Chain H deals with encryption setup, which is the most crucial action performed by ransomware.

Chain H0 is concerned with RSA key pairs generation. The *CryptAcquireContextW* function is used to acquire a handle to a key container implemented by either a cryptographic service provider (CSP) or next-generation CSP. The *szProvider* parameter specifies this information. Example:

szProvider="Microsoft Enhanced Cryptographic Provider v1.0"

The *CryptGenKey* generates a public/private key pair. The handle to the key is returned in parameter *phKey*. It has an *Algid* parameter which specifies the type of encryption algorithm being used. For example, *Algid=0xa400* represents *CALG_RSA_KEYX* as the "RSA public key exchange algorithm". The *CryptExportKey* function exports a cryptographic key pair from a CSP in a secure manner. At the receiver end, *CryptImportKey* function should be used to receive the key pair into a recipient's CSP. *CryptDestroyKey* destroys the encryption handle but not the keys. *CryptRe-*

leaseContext releases the handle of a cryptographic service provider and a key container.

Chain H1 deals with private keys and the nonce generation. Private keys can be generated using various cryptography algorithms such as AES, DES, Salsa20, etc. Most ransomware encrypts user files using a hybrid approach. Locally generated keys encrypt the victim's files, and those keys, in turn, are encrypted by the attacker's public key. Only attackers corresponding private keys can decrypt the locally generated key. The purpose of function call sequences in this chain has already been discussed before except for *CryptGenRandom*. Random IV and random keys are generated using getRandomBytes() method available via *CryptGenRandom* function.

Chain H2 encrypts the locally generated keys by using RSA public key generally obtained via the .data section. RSA private key is required from malware writers to decrypt private or local keys and decrypt encrypted user files.

Chain H3 is used for storing cryptographic key pairs. *RegCreateKeyExW* creates a specified registry keys while *RegSetValueExW* sets its data and type. Ransomware, including Gancrab family ransomware, often stores its encrypted RSA and Salsa20 keys in the registry.

Chain H4 creates a new thread to encrypt the user files. Encryption setup is in one thread, while encryption occurs in different threads. Thread creation is done using *CreateThread*. The main thread waits for all threads running on the current drive to finish by calling *WaitForMultipleObjects*. As soon as one drive is finished and all its threads end, the next drive is encrypted, and it continues until all drives have been encrypted. *WaitForSingleObject* waits for a single object to finish or for the time-out interval to elapse.

Chain I deals with file encryption.

Chain I0 deals with file encryption via Crypto API.

At first, ransomware iteratively finds the next files in a given folder to encrypt using *FindNextFileW* then writes the filename with some_unique_extension as a new filename to the buffer using *wsprintfw*. The local file names are often compared if they are not among these files: autorun.inf, ntuser.dat, iconcache.db, bootsect.bak, boot.ini, ntuser.dat.log, thumbs.db, ransom_note.html, ransom_note.txt so that it won't interfere with the system's normal functioning and should not encrypt the ransom message. *lstrcmpiW* is used to compare the discovered filename with the hardcoded list of filenames. This list or approach slightly differs among various ransomware families. The *CryptAcquireContextW* handle is called to get the *CryptoAPI* function i.e *CryptGenRandom* ready to use. Here, *CryptGenRandom* is called twice: once to generate a random 32-byte key and next time to get an 8-byte nonce. This differs among the type of symmetric encryption techniques used. *GetModuleHandleA* refers to the handle for *Advapi32* DLL and *GetProcAddress* gets the *CryptGenRandom* function. After the random key and nonce generation, the cryptographic handle is released, and virtual memory is freed.

The next *CryptAcquireContextW* is setting the cryptographic handle ready to encrypt the local private key and nonce using malware writers RSA public key. Again, the next *CryptAcquireContextW* is to encrypt the user generated RSA private key with Salsa20 keys. Here *CryptImportKey* imports the necessary keys and *CryptGetKeyParam* gets data that handle the operations of a key. *CryptEncrypt* does the real encryption of text or strings. *CryptDestroyKey* only destroys the encryption handle but not the keys. *CryptReleaseContext* releases the handle of a cryptographic service provider and a key container.

The same sequence of calls(*CryptAcquireContextW* through *CryptReleaseContext*) is observed as just before but this time to encrypt data portion from the user's file obtained via *FindNextFileW*.

CreateFileW creates a new file or opens an already existing file to overwrite its content. *ReadFile* reads the just opened file using its handle from the position specified by the file pointer. *SetFilePointerEx* moves the file pointer to the specified location, and *WriteFile* writes the given data of buffer pointer to the specified file. Finally, the *MoveFileW* function moves the file to the same or different location, but with a different filename extension, i.e., .some_extension is attached to the current filename. Again, this differs among ransomware families. Some ransomware families overwrite the filename with some random strings being generated using *CryptGenRandom* function.

Chain I1 This chain deals with encryption via CNG (Microsoft Cryptographic API Next Generation) implementation. Newer versions of ransomware, including Petya, uses this.

The *CryptAcquireContextW* is setting the cryptographic handle ready using the latest CNG. *CryptGenKey* generates symmetric keys to encrypt local files. For example, ff Algid=0x660e, then it refers to 128-bit AES keys as symmetric keys. *CryptGetKeyParam* gets data that handle the operations of a key. *PathCombineW* Concatenates two strings that represent properly formed paths into one path. It also concatenates any relative path elements example: C:/ and *. *FindFirstFileW* searches a directory for a file or subdirectory with a name that matches a specific name (or partial name if wildcards are used). The wildcard * is used to specify all files. If no files are found, and instead, a sub-directory is found, it will use *PathCombineW* as an immediate next function. Since drives generally have many directories and sub-directories, it will again call *PathCombineW*. Example: C:/ and Windows.

FindNextFileW continues a file search from a previous call to the *FindFirstFile*, *FindFirstFileEx*, or *FindFirstFileTransacted* functions. If a file is found, it combines that filename with the found path using *PathCombineW*. *PathFindExtensionW* returns the filename extension. *wsprintfW* function writes the string in .extension format to the buffer. *StrStrIW* compares this string with its given list of hardcoded extension strings. If it returns 1 meaning match found then this particular file is encrypted.

CryptEncrypt function does the real encryption of text or strings. Instead of *WriteFile* as seen in the previous chain, *FlushViewOfFile* writes the encrypted string to the disk, i.e., data obtained from a buffer of the *CryptEncrypt* function is written to the same file causing the file content to be replaced by the encrypted text. *UnmapViewOfFile* function unmaps a mapped view of a file. It removes the working set entry for each unmapped virtual page being used previously. Lastly, *CloseHandle* closes the open handle of the file that was to be encrypted.

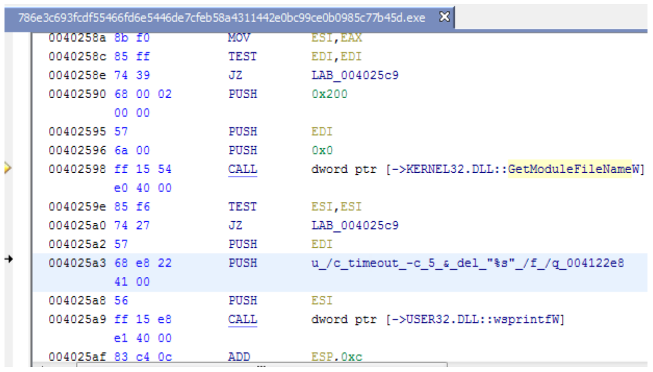
Chain J deals with creating a ransom message. *Wsprintfw* function writes some file name ransom_message.txt to buffer then creates a new file of that name and returns the handle using *CreateFileW*. *LstrlenW* gets the length of the text to be written while *WriteFile* is used to write the ransom note to the specified file. Finally, *CloseHandle* closes the filehandle given by the *CreateFileW* function.

Chain K has functions associated with network enumeration. The functions *WNetOpenEnumW, WNetEnumResourceW and WNetCloseEnum* are used in a chain for lateral movement across the network to infect more user machines.

Chain L deals with delete operations by ransomware.

Chain L0 deals with self-delete.

GetModuleFileNameW gets the malware executable location while the function *wsprintfW* writes the previously obtained command line parameter to buffer. This is also shown in figure 33 *ShellExecuteW* function via *lpFile* parameter value as *cmd.exe* executes the given command. This is a good indication of malicious activity. Why would a normal program execute a command prompt or other executable unless specified by a user? For example, web browser firefox.exe would execute its portable executable file unless we give some explicit execution commands to execute other applications.



786e3c693fcdf55466fd6e5446de7cfeb58a4311442e0bc99ce0b0985c77b45d.exe

```
0040258a 8b f0          MOV      ESI,EAX
0040258c 85 ff          TEST     EDI,EDI
0040258e 74 39          JZ       LAB_004025c9
00402590 68 00 02       PUSH     0x200
         00 00
00402595 57             PUSH     EDI
00402596 6a 00          PUSH     0x0
00402598 ff 15 54       CALL     dword ptr [->KERNEL32.DLL::GetModuleFileNameW]
         e0 40 00
0040259e 85 f6          TEST     ESI,ESI
004025a0 74 27          JZ       LAB_004025c9
004025a2 57             PUSH     EDI
004025a3 68 e8 22       PUSH     u_/c_timeout_-c_5_&_del_"%s"_/f_/q_004122e8
         41 00
004025a8 56             PUSH     ESI
004025a9 ff 15 e8       CALL     dword ptr [->USER32.DLL::wsprintfW]
         e1 40 00
004025af 83 c4 0c       ADD      ESP,0xc
```

Figure 33: Disassembled code for self deletion of ransomware binary

Table 18: Chain L: Self deletion

DLL	Function	Assembly
kernel32, user32, shell32	GetModuleFileNameW, wsprintfW, ShellExecuteW	push, mov, push-5, call, push-4, mov, call, mov, test, jz, push-3, call, test, jz, push-3, call, add, push-6, call, push, call, int

Malware can specify the number of seconds to wait. It would wait till the time elapses or until the user presses any key. The DLL, function call, and assembly used for this chain are shown in Table 18.

Chain L1 deletes the shadow copies. *GetSystemDirectoryW* gets the location to system32 director then concatenates wbem/wmic.exe to it using *lstrcatw*. Shadowcopy delete is a parameter that wmic.exe takes. It deletes shadow copies that are created when system restore points are made. These are generally backup files created by system restore operation.

Chain M deals with command and control(CC) server communication. This function chain differs among different ransomware families as some use hard-coded URL, some use domain generation algorithm, and the way to get the victim's IP address also differs. Here, the most seen common sequence is illustrated. *InternetOpenW* function opens the browser application, *InternetConnectW* opens a File Transfer Protocol (FTP) or HTTP session for a given site. Malware may use *ipv4bot.whatismyipaddress.con* to find the victim's IP address Or they could find via command prompt. In the meantime, it connects to CC server via *HttpOpenRequestW* using the handle of *InternetConnectW* function. *HttpAddRequestHeadersW* specifies the CC server. *InternetReadFile* reads the data from a handle opened by the *InternetOpenUrl*, *FtpOpenFile*, or *HttpOpenRequest* function. Finally, *InternetCloseHandle* closes the internet handle.

7.4 Experiments

The focus of this work was an analysis of crypto-ransomware and proposing an efficient detection framework. Accordingly, the need for recent crypto-ransomware binaries was fulfilled by VirusTotal [45] whom I found open to support the research community for malware/ransomware study. I collected 2600 malware samples, including 550 crypto ransomware from VirusTotal [45]. The malware sample distribution and the size range are shown in table 19. Also, 540 benign application samples were collected from Windows 7/10 OS and open-source applications. It includes normal to advance programs such as *cmd.exe*, *explorer.exe*, *bitlocker.exe*, *firefox.exe*, *openssl.exe*, *taskkill.exe*, *ssh-agent.exe*, *winScp.exe*, *ssh-keygen.exe* and so on. Among them, there are certain samples such as *Openssl*, *WinScp*, and *BitLocker* which uses cryptographic and communication operations which are also the properties seen in the ransomware samples. Normal binaries size varied from 16.4 KB to 36.3 MB which resembled the size of ransomware binaries.

Experimental protocols and evaluation Measures are the same as discussed in chap-

Table 19: Malware families and number of samples (with sizes) used

Malware Type	No. of Samples	Size range
crypto Ransomware	550	17.1KB - 31.4MB
Adware	524	8.1KB - 11.7MB
Backdoor	498	22.4KB - 21MB
Trojan	513	5.1KB - 25.5 MB
Worm	515	51KB - 17.5 MB

ter 5. The addition is the use of a novel approach of calculating the ransomware profiling chain ratio as discussed in the chain validator section. For association rule mining, the minimum support threshold is set to 2 and the confidence threshold to 0.8, but only association rules with a score of 1 are shown in section 1.6.

7.4.1 Result analysis

The experiment done with combined multi-level features with term frequency for two classes (ransomware and benign samples) is shown in table 20. SVM and Adaboost with J48 reported the second-highest accuracy of 99.54% and lowest false positive rate of 0.005. J48 achieved the second-highest accuracy of 99.26% and a false positive rate of 0.007. A Neural network with two hidden layers, ReLu activation function, and L1-L2 regularizers, was used as an unsupervised learning technique, which achieved the highest accuracy of 99.69% and a false positive rate 0.005. This achieved improved detection due to improved optimization during training phases. Another experiment done with TF and tri-gram TF-IDF for the same two classes as above is shown in the table 21. This table shows some improvements to the previous one due to the use of the tri-gram TF-IDF approach. SVM reported the highest accuracy with 99.72% and the lowest false positive rate of 0.003. SVM performed better than other algorithms because SVM works more effectively to handle high dimensional spaces, and there is less overlapping among target classes.

The improved accuracy seen here than the similar previous approaches [100, 102] is due to the hybrid reverse engineering techniques used, which builds a unique feature set.

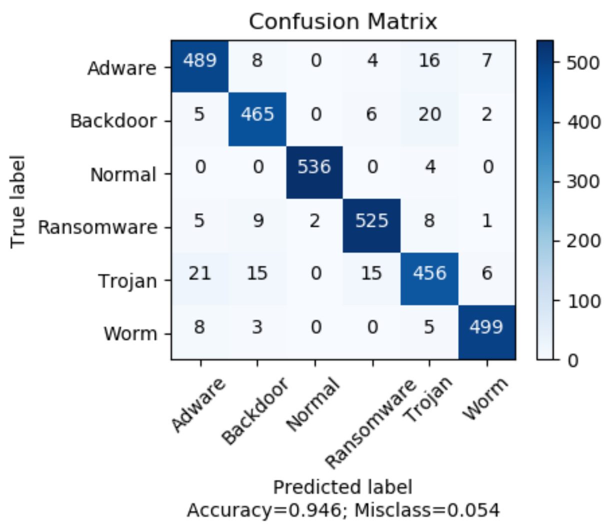


Figure 34: Confusion matrix for multi-class malware using Adaboost with J48

Table 20: Machine learning algorithms' evaluation for multi-level combined approach with TF for two classes (Ransomware and benign samples)

Machine learning Classifier	TPR	FPR	Precision	Recall	F-measure	Accuracy (%)
Logistic Regression	0.992	0.008	0.992	0.992	0.992	99.17
Support Vector Machine	0.995	0.005	0.995	0.995	0.995	99.54
Random Forest(RF)	0.990	0.010	0.990	0.990	0.990	98.99
J48	0.993	0.007	0.993	0.993	0.993	99.26
Adaboost with RF	0.987	0.013	0.987	0.987	0.987	98.71
Adaboost with J48	0.995	0.005	0.995	0.995	0.995	99.54

Even the experiments done with multiple malware classes show promising results with an overall accuracy of 94.6% with the J48 machine learning classifier as shown in table 22. The confusion matrix for the same experiment is shown at figure 34. The table and confusion matrix shows the high true positive and low false-positive rates for various malware classes, including the normal class. Here, the Adaboost with J48 outperformed SVM because this experiment included more than 3000

Table 21: Machine learning algorithms' evaluation for multi-level combined approach with TF and Tri-gram TF-IDFs for two classes (Ransomware and benign samples)

Machine learning Classifier	TPR	FPR	Precision	Recall	F-measure	Accuracy (%)
Logistic Regression	0.996	0.004	0.996	0.996	0.996	99.63
Support Vector Machine	0.997	0.003	0.997	0.997	0.997	99.72
Random Forest(RF)	0.990	0.010	0.990	0.990	0.990	98.99
J48	0.996	0.004	0.996	0.996	0.996	99.63
Adaboost with RF	0.992	0.008	0.992	0.992	0.992	99.17
Adaboost with J48	0.995	0.005	0.995	0.995	0.995	99.54

Table 22: Machine learning algorithms' evaluation for multi-level combined approach with TF for multi-class malwares using Adaboost with J48 (Malware families and benign samples; accuracy: 94.58%)

TPR	FPR	Precision	Recall	F-measure	Class
0.933	0.015	0.926	0.933	0.930	Adware
0.934	0.013	0.930	0.934	0.932	Backdoor
0.993	0.001	0.996	0.993	0.994	Normal
0.955	0.010	0.955	0.955	0.955	Ransomware
0.889	0.020	0.896	0.889	0.892	Trojan
0.969	0.006	0.969	0.969	0.969	Worm

samples and had more overlapping than the previous dataset with two classes. J48 decreases the complexity of the final classifier by effective pruning and thus handling the problem of overfitting while the Adaboost further boosts its classification tasks. TPR and FPR score inspection for multiple classes, including normal samples, show our approach's performance is not degraded even in the case of multiple malware classes.

Table 23 shows the ransomware profiling chain ratio, including other scores for six crypto-ransomware family samples. A chain score close to 1 is expected to determine the sample as ransomware. A score equal to one is a perfect score. The threshold chain ratio score is chosen as 0.8 for RPCR. The table shows all the samples to achieve this score and can be determined as a ransomware sample.

Table 23: RPCR calculation for different crypto ransomware families

Ransomware family	DCR	FCR	ACR	RPCR
Locky	0.78	0.94	0.81	0.84
TeslaCrypt	0.81	0.95	0.85	0.87
GandCrab	0.88	0.93	0.89	0.9
CryptoWall	0.85	0.91	0.81	0.85
Cerber	0.88	0.95	0.88	0.903
Petya	0.78	0.81	0.83	0.806

Table 24: Comparison based on various factors of this framework against existing approaches

Analysis/ Reference	Feature	Classification	Accuracy	FPR	Resilience to obfuscation
Static [86]	Opcode	CNN	97%	NA	Low
Dynamic [81]	API	RF, DNN	97.3%	0.048	Low
Dynamic [78]	API	Frequency analysis	NA	NA	Low
Dynamic [53]	HPC event traces	ANN, FFT	NA	NA	Low
Dynamic [56]	Dlls, system resources	Supervised, unsupervised	75.01%	NA	Low
Dynamic [107]	API	Deep learning	95.96%	0.059	Low
Dynamic [60]	API	Supervised	98.65%	NA	Low
Hash based [72]	Data entropy	NA	99.24%	0.0049	Medium
Hybrid [Mine]	Dll, function, assembly	Supervised, unsupervised	99.72%	0.001	High

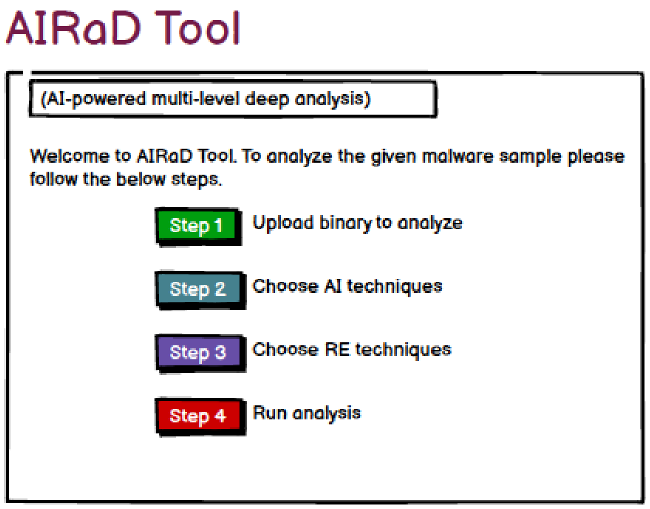
7.4.2 Comparison with others

Many works have been done using API or opcode analysis for ransomware detection that leverages portable executable file format. However, I have compared the recent and notable ones in Table 24. This table shows different features used, classification techniques, resilience to obfuscation, accuracy, and false-positive rate of the proposed model. The description of each work is already discussed in the literature review section. Here, I summarize the table and note the differences with our work. Most of the compared works deal with dynamic analysis and a few with static and hash-based analysis. But our approach deals with hybrid analysis which is a combination of both static and dynamic analysis techniques. The advantage of hybrid analysis is its ability to capture features and behaviors without executing the binaries. Static analysis, though it can achieve a high code coverage, misses the actual run-time conduct. However, there are few downsides of dynamic analysis like missing parameter values, and we may not capture the essential behavior or miss certain notable behaviors of ransomware executables. The significance of the hybrid approach, along with the feature capture at Dll, function call, and assembly level, can be observed by our work which also achieved high accuracy and lowest false positive rate, among other results. Another notable difference is how resilient your approach is to obfuscation. Most related work has low resilience to obfuscation and fails to explain whether their model can detect obfuscated binaries. My approach has high resilience to obfuscation as I try to capture malware at three levels; even if we miss the top Dll level, we can capture at either function call level or the lowest assembly level. My unique hybrid analysis method using advanced tools, including intel's PIN and NSA's Ghidra, allows creating a rich feature set for machine learning training models. Thus, I see a high potential for my work compared to other related works.

7.5 Designing a prototype system AIRAD

I designed a prototype system based on the proposed HMLP approach described earlier.

The prototype that I developed is referred to as AIRaD (AI-powered Ransomware Detection) tool. This tool is in the development process, and I plan to make some of its components open-source. Figure 35 is a welcome screen that shows the significant four steps required for the malware analysis process. Step 1 includes uploading a binary file for analysis. Step 2 allows us to choose among the available AI techniques. Similarly, step 3 will enable us to choose among the available reverse engineering techniques. For steps 2 and 3, if we are unsure about which methods to choose, one can let the system select the best ways to guarantee the best perfor-

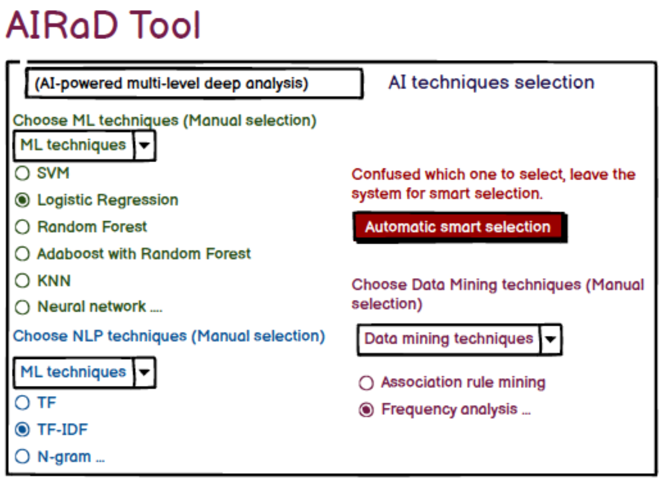


AIRaD Tool

(AI-powered multi-level deep analysis)

Welcome to AIRaD Tool. To analyze the given malware sample please follow the below steps.

Figure 35: Welcome screen of AIRaD tool



AIRaD Tool

(AI-powered multi-level deep analysis)

AI techniques selection

Choose ML techniques (Manual selection)

ML techniques

SVM [✓ Logistic Regression] Random Forest Adaboost with Random Forest KNN Neural network ...

Choose NLP techniques (Manual selection)

ML techniques

TF [✓ TF-IDF] N-gram ...

Confused which one to select, leave the system for smart selection.

Choose Data Mining techniques (Manual selection)

Data mining techniques

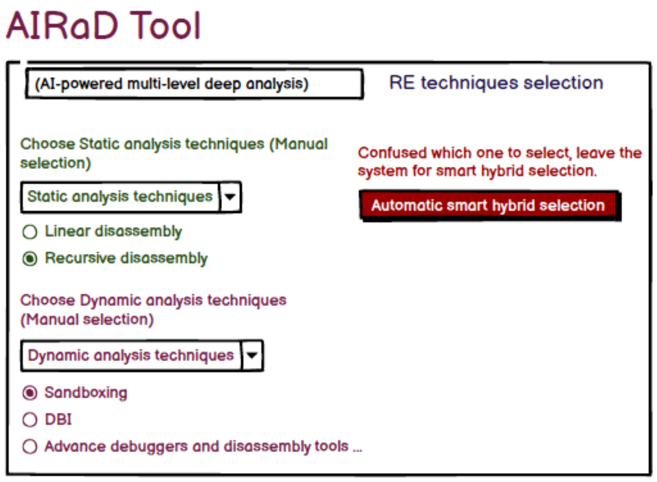
Association rule mining [✓ Frequency analysis ...]

Figure 36: AI techniques selection for AIRaD tool

mance.

Figure 36 shows the interface to select available AI techniques. A user is asked to select among available machine learning techniques. Here, various supervised and unsupervised machine learning techniques are available to choose from. Similarly, users can choose available NLP and data mining techniques. The automatic smart selection button allows the user to let the system decide on optimal techniques.

Figure 37 allows user to select available reverse engineering techniques. A user can select either linear disassembly techniques such as Objectdump or recursive dis-


AIRaD Tool
(AI-powered multi-level deep analysis)
RE techniques selection
Automatic smart hybrid selection

Choose Static analysis techniques (Manual selection)

Static analysis techniques

○ Linear disassembly
[✓ Recursive disassembly]

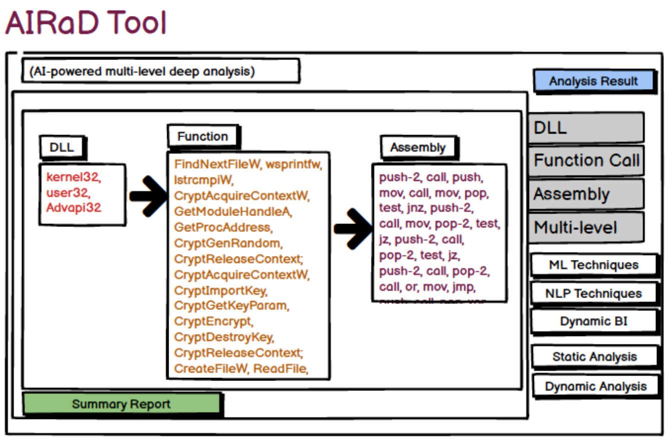
Choose Dynamic analysis techniques (Manual selection)

Dynamic analysis techniques

[✓ Sandboxing]
○ DBI
○ Advance debuggers and disassembly tools ...

Confused which one to select, leave the system for smart hybrid selection.

Figure 37: RE techniques selection for AIRaD tool



DLL

kernel32, user32, Advapi32

Function

FindNextFileW, wsprintfw, lstrcmpiW, CryptAcquireContextW, GetModuleHandleA, GetProcAddress, CryptGenRandom, CryptReleaseContext; CryptAcquireContextW, CryptImportKey, CryptGetKeyParam, CryptEncrypt, CryptDestroyKey, CryptReleaseContext; CreateFileW, ReadFile,

Assembly

push-2, call, push, mov, call, mov, pop, test, jnz, push-2, call, mov, pop-2, test, jz, push-2, call, pop-2, test, jz, push-2, call, pop-2, call, or, mov, jmp,

Figure 38: Snapshot of AIRaD tool with associated analysis components and summary report

assembly provided by Capstone tool [10]. For dynamic analysis, the user can select sandboxing API analysis, dynamic binary instrumentation provided by Intel's PIN [32] or other advanced debuggers and disassemblers such as IDA Pro [25] and NSA's Ghidra [21]. Users can choose the techniques themselves or allow the system for automatic smart hybrid selection.

Figure 38 shows one of the output interfaces of our tool. It shows the multi-level mapping for file encryption activity. The main box shows DLL, function call, and assembly components with an arrow pointing from DLL to the assembly level. This association among these three levels is significant to recognize ransomware-specific behavior and create unique signatures. The upper part of the rightmost column shows buttons to choose either one level of analysis or multi-level analysis. The bottom portion of it shows buttons for machine learning techniques, NLP techniques, Dynamic binary instrumentation, and Static and dynamic analysis approaches used in our tool. The user can download the summary report for record-keeping and further analysis. This tool is built upon the foundations provided by our proposed architecture and can be considered as an explanatory AI tool as it identifies the signature generating features.

7.6 Summary

In this work, I performed a deep forensic analysis of crypto-ransomware using hybrid multi-level profiling. I adopted a unique approach of behavioral chaining along with association rule mining and AI techniques. Hybrid multi-level inspection at DLL, function call, and assembly revealed unique behavioral chains that help create unique ransomware signatures and a distinguishing dataset for the machine learning model. This approach is validated with experiments where I achieved high accuracy with low false positives. Results show that one machine learning algorithm achieved the highest accuracy of 99.72% and a false positive rate of 0.003 with two class datasets. Experiments done at multi-class malware families also revealed a reasonable accuracy rate (94.6%) with a very low false-positive rate of 0.001. Ransomware behavioral profiling chain ratio is a novel approach to identify ransomware binary, and it shows significant detection accuracy.

8 Malware Analytics: Review of Data Mining, Machine Learning, and Big Data Perspectives

Recent advances in cyber technologies have made human life's more accessible, but it may lead to a high cost in terms of economic, psychological, or reputation damage. For instance, these damages may be caused by variants of malware propagated in a hidden and mostly untraceable way. Malware analytics deals with the approaches and techniques utilized to generate the distinguishing characteristics of the malware for robust cyber defenses. This work presents the current status of the malware research, challenges, and methods used to overcome those challenges using data mining, machine learning, and big data perspectives. Because of their extensive computation value, I have considered these three perspectives, mostly fused to solve a wide range of problems from security to medical, finance, and industry. These domains as an independent technique and their interrelationships depend on the nature of the dataset considered. I have also proposed a framework to overcome the challenges and open issues prevalent in malware analytics. With the simplified presentation of the most vital approaches of malware analytics, it is hoped that this work will help the inspiring researcher or a newbie in the security field explore more and budding engineers to choose malware analysis as their field of study. Specifically, analysis of state-of-the-art approaches with evaluation, pros and cons discussion, and the current challenges and future directions will empower all the malware enthusiasts.

8.1 Challenges

Malware analysis has always been a challenging field. The main reason is the tug of war between the malware writers and attack defenders. Both entities are smart. Malware writers have the upper hand as they also write the code to obfuscate the program or hide its presence and make disassembling very difficult for the defenders. Some of the significant challenges are described below.

1. Most of the anti-virus engines are based on signature-based detection. They particularly match the hash value of the executable or file in consideration and match it with the hash list in their database. Some engines also see some known listed patterns as a signature. This signature-based detection approach is a bottleneck for malware detection and prevention as malware writers can easily bypass the signature-based detection solutions.

2. Code obfuscation is the technique to transform the original program code by changing its flow or structure to make it difficult to debug or reverse-engineer. This comes with one benefit that helps preserve the original code, its authenticity and prevent unauthorized tempering. However, its downside is that malware writers do this so that reverse engineering is complex and detection technique is challenging to formulate.

3. Petabytes and exabytes of information are being transferred between network nodes every second, making rooms for hackers to enter into. There is a need for intelligent monitoring of offline and live traffic data to built an alert system, which will alarm if there is any anomaly in the network or system. Moreover, the traditional way of creating a detection system is not capable of processing big data. The detection system that uses various data mining and machine learning techniques should handle and process big data.

4. Data coming from different sources and in different formats is another challenge. Many researchers and defense solution companies handle this, but many are still unaware of handling it intelligently. Owing to the heterogeneous living environment around us, the data sources are heterogeneous and come in different forms and formats. This can be a challenge to a detection system

since it needs to handle this so that there is no obstruction in the detection process.

In this work, I discuss how each of these challenges is overcome and discuss the open issues that need to be resolved.

8.2 Data Mining Perspective

Data mining is the process of finding hidden useful features or patterns from the given dataset or sample set. The process involves all the mining steps starting from the raw data or sample to get useful concluding features or information. It is a multi-disciplinary approach that uses technologies such as probabilistic theory, machine learning, artificial intelligence, and database. Data mining is sometimes also termed knowledge discovery or knowledge extraction.

Data for data mining is obtained from the data warehouse, database systems such as relational or object-oriented, legacy, multimedia, or real-time database systems. But for malware analytics, data mining involves obtaining raw data as an initial feature set from malware samples. So, this starts with static or dynamic analysis of the malware executable to receive the initial raw features, which will be mined to get the useful distinguishing patterns to make the detection and classification process effective. The general steps in the data mining process are shown in Figure 39 and described briefly below.

Malware samples are collected from various sources such as VirusTotal and TheZoo open-source repository. The Raw feature extractor component uses either static, dynamic, or hybrid techniques to extract raw malware features. These may be application programming interfaces (APIs), function calls, system calls, opcode, assembly sequences, or raw information from Portable executable (PE) file format. This raw information is pre-processed to remove redundant or unnecessary data, remove outliers, normalize or smooth the data. The next step is to get the analyz-

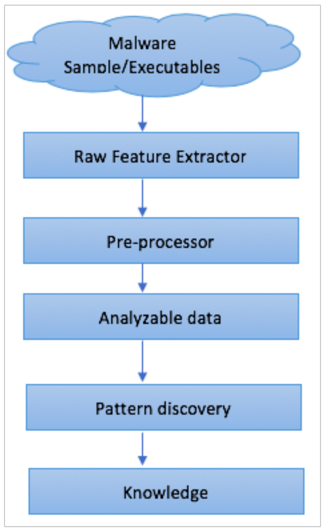


Figure 39: Steps in Data mining for Malware analytics

able data where the data is prepared either in a CSV file, JSON, or other formats. The obtained feature dataset is passed to the pattern discovery phase, where different classification and clustering techniques are applied to get valuable patterns or knowledge.

8.3 Machine Learning Perspective

Machine learning is the technique that gives the computer the ability to learn and predict the output based on the learned patterns. Machine learning is achieved by various machine learning algorithms where most of them use different probabilistic approaches. These days machine learning has been heavily used in almost every sector of our life. The acceptable prediction rate and its easy operation make complex tasks easier to implement. More and more people nowadays rely on machine learning. Machine learning has proved beneficial for malware analytics and has been used by security researchers and anti-virus companies. The generalized operation of machine learning steps for malware analytics is shown in Figure 40. The machine learning perspective of malware analytics involves two major compo-

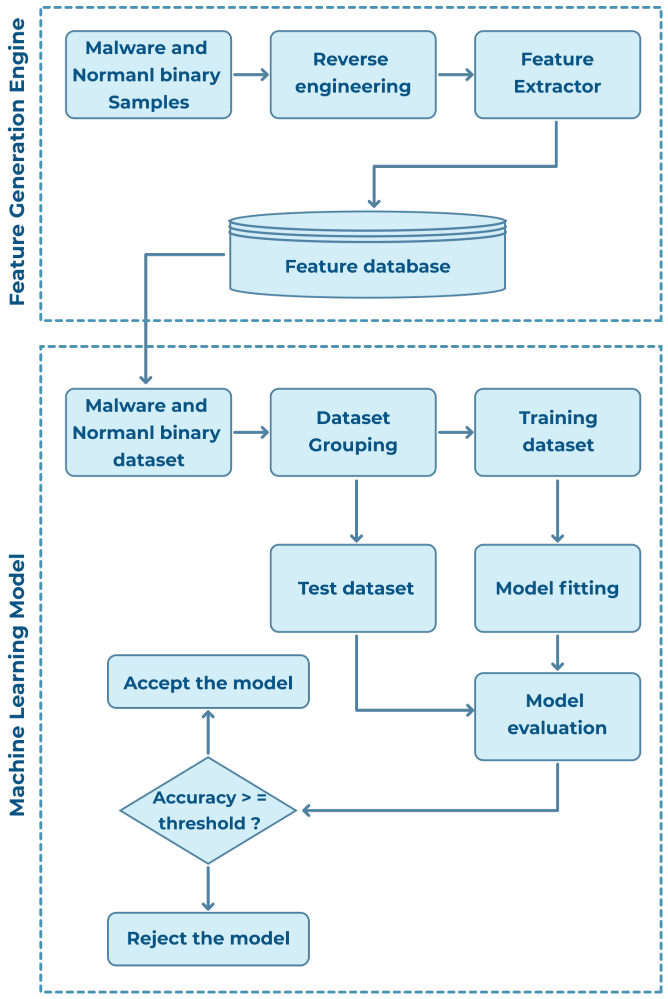


Figure 40: Steps in Machine learning for Malware analytics

nents: feature generation engine and machine learning model. Feature generation engine starts with the collection of malware and normal sample. I then apply various reverse engineering techniques such as binary analysis and debugging using custom program codes (researcher's code using Linux tools such as ObjDump or other available libraries for other operating systems), open sources (PE parser, Angr) or commercial tools (IdaPro).

Reverse engineering is the backward process of trying to achieve what the program is intended to do and to know the structure of the program. Malware writers make the task of reverse engineering difficult by applying various obfuscation techniques. The result of the reverse engineering step gives some raw data, which is pre-processed by the feature extractor component. This gives the clean feature dataset.

The second but vital component is the machine learning model, whose input feed is the malware and normal binary dataset obtained from phase 1. The dataset is divided into two subsets as training and test dataset. This division is based on the program analyst or research team choice. K-fold cross-validation and 60 to 40 percent split is most commonly used. The training dataset is fit into the machine learning model. Machine learning algorithms implemented here may be supervised or unsupervised depending upon the nature of the dataset. The model is then evaluated with the test dataset. The accuracy should be above the defined threshold value. The Machine learning model is accepted if the accuracy is above the threshold value. Otherwise, the experiment is re-run with tuning parameters, remodeling or changing approach used, or algorithms.

8.4 Big Data Perspective

The world is growing with a massive quantity of data termed big data. Data often grows with increased services and resources used by different individuals and entities in an organization or a company. A social networking site, blog site, customer browsing history, e-commerce tracking, network traffic, financial transaction, medical data all add up every second, making huge tons of data. This comes up with the data management challenge and opens the door for hackers and other adversaries.

Every individual, company, or organization wants to reduce or prevent the damage caused by malware attacks. They want to detect and thwart malware attacks as soon as possible. This would be handy with small data sources, but we have a huge unavoidable amount of data. Primitive techniques with limited resources and processing capabilities are not able to handle big data. Apache Hadoop and Apache Spark have made the task of big data analysis convenient and efficient. Among these two big data distributed frameworks, researchers mostly use Apache Spark as it supports the machine learning model and real-time processing for their malware analytics job.

8.4.1 Apache Hadoop

The first distributed computing big data framework operates in two layers: Hadoop distributed file system (HDFS) and MapReduce layer. HDFS is the storage layer responsible for storing data in multiple nodes in a distributed fashion. At the same time, MapReduce is responsible for Map and Reduce functions to process big data in Hadoop clusters. Figure 41 shows the layers with name node, data nodes, and job trackers.

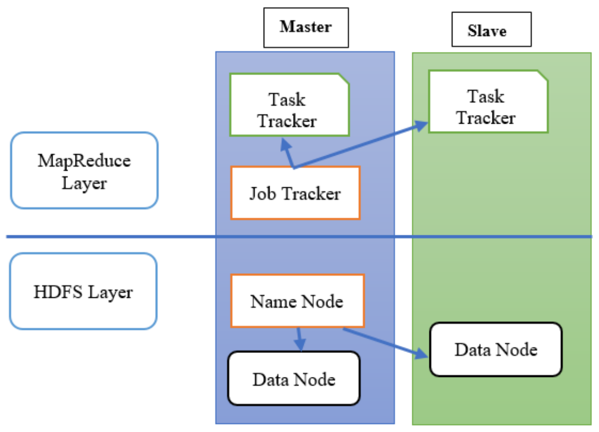


Figure 41: Basic architecture of Apache Hadoop framework

Apache Hadoop is generally used to batch large datasets where we do not need the intermediate solution. One downside of using Hadoop is that its slow performance due to input-output disk latency.

8.4.2 Apache Spark

Apache Spark is the popular distributed big data processing framework, which consists of Spark core and a set of libraries. It has libraries for SQL queries, streaming, machine learning, and graph processing. Spark core is responsible for managing our submitted job, i.e., it manages the handling of data, processing, execution, and result delivery. Different components of the Apache Spark framework are shown in Figure 42.

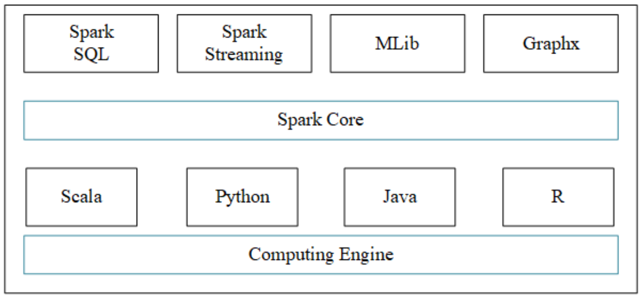


Figure 42: Basic architecture of Apache Spark framework

Apache spark has many advantages over Apache Hadoop. For example, it can process batch data and real-time data processing, unlike only batch data by Apache Hadoop. In terms of file compatibility, Spark depends on the data sources and file formats supported by Hadoop. Spark processing is faster than Hadoop due to its Resilient Distributed Dataset (RDD), which supports in-memory processing computation. The state of memory is stored as an object across the jobs, and the object is shareable between those jobs [6]. Apache Spark supports the following programming languages: Java, Scala, Python, and R. Whereas Hadoop supports Java primarily, but languages like C, C++, Ruby, Python, Perl, Groovy are also supported.

8.5 Detection Approach

In this section, I discuss the general detection approach for malware analytics using big data frameworks. Figure 43 shows the basic building blocks for the detection approach.

Table 25: Summary of recent works for malware analytics (DM: Data mining, ML: Machine learning, BD: Big data,)

Study	Perspective	Feature Selection	Classification Schemes
Tariq et al., 2013 [93]	DM + BD	Network traffic, web transactions	ETL, Association rule
Wen et al., 2014 [116]	ML + BD	API's n-gram	SVM
Arp et al., 2014 [57]	ML	APIs and Permissions	SVM
Joshua et al., 2015 [104]	ML	Contextual byte and PE features	Deep Neural network
Daniel et al., 2016 [71]	BD	N-grams sequence for decompiled byte information	Smith-Waterman algorithm
Chia et al., 2016 [65]	ML + BD	PE file features	Random forest, SVM, Neural network
Kolosnjaji et al., 2016 [89]	ML	N-gram modeling of system call sequences	Convolutional and recurrent neural network
Hardy et al., 2016 [79]	ML	API calls	Deep learning using SAEs
Fan et al., 2016 [73]	DM	Assembly instructions	All-Nearest-Neighbor
Ramkumar et al., 2017 [97]	DM + ML	Hex-code and assembly n-grams, dll and assembly instructions	Random forest, SVM
Mozammel et al., 2017 [67]	DM + ML	N-gram of samples and APIs with PCA	Various Supervised classifiers
Hou et al., 2017 [80]	DM + ML	API calls for information network	Hindroid(Multi-kernel learning)
Anderson et al., 2017 [54]	ML	PE file features	Reinforcement learning
Vinayakumar et al., 2018 [114]	ML	PE file features	Deep Neural network, classical ML
Shankar et al., 2018 [106]	DM + BD	Web traffic, location, permission, etc.	Signature matching
Amin et al., 2018 [58]	DM + ML	N-grams of opcodes	Deep eigenspace learning
Rafal et al., 2018 [90]	ML + BD	Network features(IPs, Packet, etc.)	Distributed machine learning(EL and RF)
Cui et al., 2018 [70]	DM + ML	Code to grayscale image	Deep learning, CNN, Bat algorithm
Li et al., 2018 [91]	DM + ML	Permission usage	SVM, Association rules
Bacci et al., 2018 [59]	ML	Opcodes, System calls	SVM
Poudyal et al., 2018 [102]	ML	Assembly instructions, Dlls	Various supervised classifiers
Kolosnjaji et al., 2018 [88]	ML	Byte code	Deep neural network
Poudyal et al., 2019 [100]	ML + BD	Dll, Function call and Assembly	Various supervised classifiers
Wang et al., 2019 [115]	ML	API call sequences	Deep autoencoder, CNN
Yuxin et al., 2019 [119]	ML	Opcode of PE file	Deep belief network (DBN)
Xiao et al., 2019 [117]	ML	API calls	Deep Learning of behavior graphs

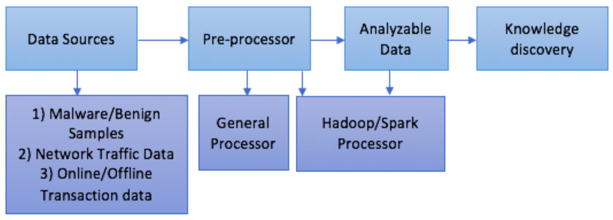


Figure 43: Malware analytics using Bigdata frameworks

Data sources can be malware/benign executable files, network traffic data, or online/offline transaction data from where we want to detect an anomaly or unusual pattern. The pre-processor component is the most powerful component of this architecture. The data sources go into some pre-processing tasks using normal computing processing capabilities or solely done using Hadoop/Spark frameworks. The result of this is clean and analyzable data where we apply various machine learning or data mining algorithms to find a useful pattern, also termed as knowledge discovery.

8.6 Current Research techniques

In this section, I discuss the solutions proposed by various researchers. Each of them has tried to address one or more challenges that I discussed in section 8.1. Table 25 shows the comprehensive study of the techniques and perspective used. However, some of the works are discussed in more detail below.

Tariq et al. [93] have proposed the Big Data Analytics (BDA) process, which assists network managers in monitoring and surveillance of real-time network streams and real-time detection of malicious and suspicious patterns. BDA helps by detecting and predicting suspicious sources and destinations, detect and predict abnormal user access patterns, abnormal or sudden configuration changes, abnormal usage patterns, abnormal access time, or transaction amount. The three complexities of big data are stated as volume: terabytes or exabytes of data; variety: co-existence of unstructured, semi-structured, and structured data; velocity: the rate at which the data is generated. BDA tries to address these challenges using different data mining techniques such as predictive analytics, cluster analysis, and association rule mining. Data are classified as either passive or active. To detect the cyberattack, they monitor the following dimensions: network traffic, web transactions, network servers, network source, and user credential.

Ramkumar et al. [97] have explained different types of malware datasets they collected from different sources. They have used data mining and machine learning based on N-grams, Apache Hadoop, and Apache Spark for malware classification and detection. They have used byte 4-grams, assembly 4-grams, Dll imports, assembly frequencies, Random forest, SVM, and Information gain. They have 95,608,217 byte 4-grams, 419,888 assembly 4-grams, 26,785 Dll imports, and 82 assembly instructions. The experiment done in Spark took 16.42 minutes, while Hadoop took 29.37 minutes to run the classification/detection experiment. The average accuracy they got was 96.32% for various parameters run.

Daniel et al. [71] have used the Smith-Waterman algorithm and Apache Spark to find malware sequence alignment so that it can be used to identify malware files faster than performing sequence alignment on two complete files. The Smith-Waterman algorithm is stemmed from the Needleman_Wunsch algorithm. The Smith-Waterman algorithm compares segments of all possible lengths and finds the most optimal sequence alignment. Distance matrix alignment is used to find the longest common sequence of bytes in two given malware samples based on similarity scores. N-grams of byte sequences are considered. An alignment between two files can take up to 10 hours, where each file has a size of 20MB. The experiment took 51 minutes for Apache spark in a Standalone cluster mode. There was not much performance gain as they used a single cluster.

Chia et al. [65] have used the dataset of malware and benign files from the static and dynamic analysis done by virustotal. Their main goal is to find the minimum number of features for training the machine learning model so that the training efficiency of the algorithm is improved significantly in polynomial time with the number of features. Spark and Torch were used along with the Support vector machines, Random Forest, and Neural network classifiers. The features considered were a number of imports, DNS_host, PE overlay size, CALLS, Lang code, etc. Due to the large dimensionality, they used Spark and MLib for feature selection. ChiSqSelector was applied to get 10% of the most relevant features. They reduced the matrix containing 68,800 features with 9448 observations and of size 2.2 GB. Creating dataset and training took 16 minutes; NN required 255 iterations in 13 minutes. With SVM and nine features selected, the accuracy reached 99.24%.

Wen et al. [116] have proposed DroidDolphin using APImonitor, SVM, and Hadoop clusters. DroidDolphin uses a dynamic malware analysis approach using big data analysis and machine learning to detect malicious Android executables. API monitor tool was used to capture API sequences for Android-based applications, and an N-gram model was used to generate features to feed into the Support vector machine (56354 number of dimensions as features). They parallelized the work on 32 AVDs (Android virtual devices) using Spark. It took up to 5 minutes for emulators to run the APKs (Application programming) and 24 hours to process 2700 applications using 32 AVDs (Android application packages). They used 32,000 benign and the same number of malware samples and got an accuracy of 86.1%.
Rafal et al. [90] have used distributed machine learning for Botnet activity detection. They proposed using and implementing cost-sensitive distributed machine learning through distributed Extreme Learning Machines (ELM), distributed random forest, and distributed random boosted-trees. Data were analyzed in NetFlow (ports, protocols, IPs, packets, etc., were considered). To make the analysis efficient and scalable, they have proposed to collect the NetFlow data in an HDFS system and Map-Reduce. Experiments were run using 1 and 8 Apache spark nodes. By using spark clusters, they were able to improve the training process by a factor of 4.5 for ELM and 3.7 for Random forest and gradient boosting trees. Overall classification efficiency was considered better for ELM than others for various scenarios, so it is proposed as an efficient to use.

Amin et al. [58] have proposed a malware detection approach for IoT devices. Opcode sequences were considered for feature selection which was fed to the Eigenspace learning method for malware classification. They also claim that their approach can minimize the junk code insertion attack. Detection accuracy was found to be 98.37%.
Hou et al. [80] has proposed a smart Android malware detection system by analyz-

ing not only the API sequences but by studying their close interrelationship to get some meaningful insights. They have used the meta path to get similarity scores for apps and used multi-kernel learning. The data mining tasks are achieved through unzipper, feature extractor, and multi-kernel learner. SVM is used to evaluate the experimental results.

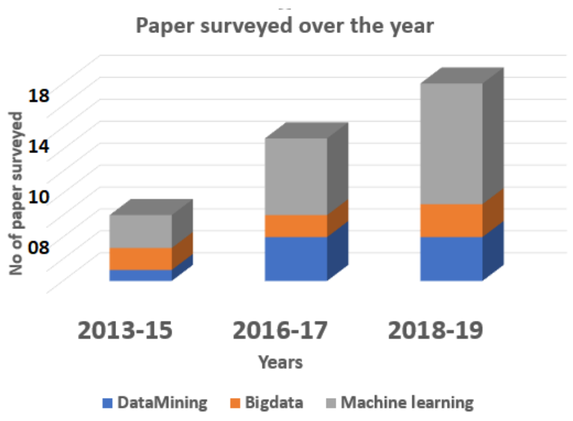


Figure 44: Papers surveyed over a year

From this work, I found that the current techniques are more focused on machine learning, particularly deep learning methods for malware detection (See Table 25 and bar graph in Figure 44). Figure 44 gives an overall trend of use of three perspectives where there is a distinctive increase of machine learning techniques in recent years.

8.7 Open issues

The research approaches discussed in the previous sections tried to solve the challenge of malware detection by signature as a bottleneck. Handling of large features and petabytes of the dataset was addressed using either Hadoop or Spark distributed system. Handling code obfuscation is a continuous challenge and depends on smart feature selection techniques. Handling different data formats deals with smart pre-processing in a distributed environment. Most of the research is focused on static analysis of malware. Only considering static approaches will not be effective as malware writers modify their bad purpose code to bypass the detection tools and make it difficult for disassembly or static feature selection. The dynamic feature captures the running behavior of the malware. However, it also comes with some cost as most malware may not run in a virtual box, sandbox, or other dynamic analysis environments. The malware code has been written so that if any of the environments mentioned above is detected, the malware may not execute at all or may exhibit a different behavior to hide its true intention and make the job of defenders too difficult. A combination of both static and dynamic features is considered a good balance and can give better accuracy while doing feature analysis. Another open issue, which is very challenging to solve, is the malware behavior at different OSI (Open Systems Interconnection model) stacks and different levels of program execution, mainly assembly, function call, and system call. Many research has been successful in analyzing the behavior in one or more levels, but not all. Below I have discussed the open issues and possible solutions categorized in three perspectives.

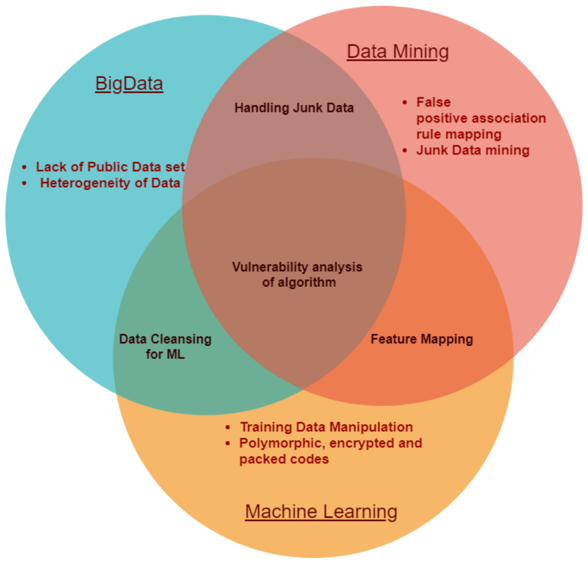

BigData
Data Mining
Handling Junk Data
• False positive association rule mapping
• Junk Data mining
• Lack of Public Data set
• Heterogeneity of Data
Vulnerability analysis of algorithm
Data Cleansing for ML
Feature Mapping
• Training Data Manipulation
• Polymorphic, encrypted and packed codes
Machine Learning

Figure 45: Open issues in three perspectives

8.7.1 Data Mining

A signature or heuristic-based malware detection is easily thwarted by badly smart malware writers because they apply different techniques to hide or obfuscate the code. Due to this, the data mining algorithms become prey to false-positive association rules mapping. Feature mining is then considered junk data mining. For example, in assembly code analysis of malware portable executable files, we get a certain sequence of assembly instruction with high frequency but is irrelevant in knowledge discovery. The proposed solution to these problems is to use Cognitive data mining. I recommend using context-sensitive association rule generation and adopting validation of the knowledge discovery process.

8.7.2 Machine Learning

Machine learning algorithms become a victim of training data manipulation by adversaries. The portion of the training data may be deleted, modified, or added with false cases. This will affect the learning process, and a malware sample may be considered a normal sample. Apart from this, the polymorphic codes, encrypted and packed codes make it difficult for the security or malware analyst to design and train a good machine learning model. The proposed solution to these open issues is to use deep monitoring of the training dataset, which consists of hashing and timestamping the dataset to maintain its integrity. On-time behavioral monitoring is suggested to defend the polymorphic behavior of packed or encrypted codes.

8.7.3 Big Data

Terabytes of feature processing which includes log or network monitoring data and features obtained from the reverse engineering process, is a challenging task. This has been effectively handled by Big data frameworks like Hadoop and Spark. But, the heterogeneity of data makes the handling and processing of big data challenging. Moreover, there is a lack of publicly available datasets for Big data processing. Security vulnerabilities of big data framed machine learning algorithms for malware datasets are yet to be studied. However, various commercial products are available for real-time and offline big data processing, which has made the computation effective. A feature-based data cleansing process is suggested to make categorical homogeneous data.

Figure 45 gives a Venn-diagram representation of major open issues in our mentioned three perspectives.

8.8 Proposed Framework

The previously discussed open issues in all three perspectives motivated us to propose a Malware analytics framework which I strongly believe will help in mitigating the current challenging issues. Figure 46 shows different steps and interactions of the proposed framework [98]. A brief discussion of its components follows below.

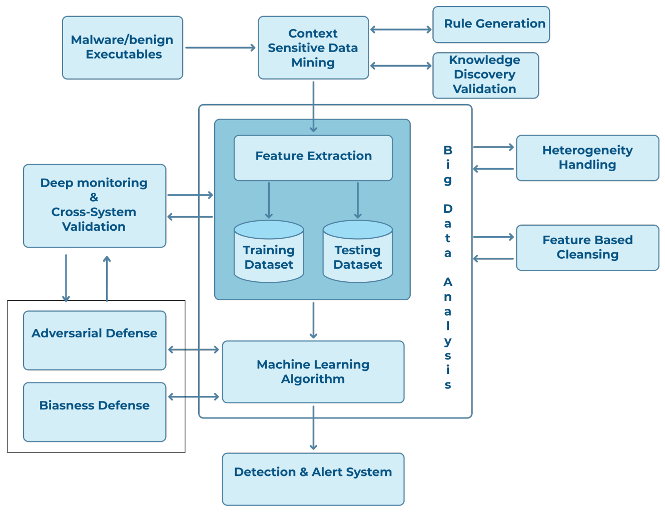


Figure 46: A Proposed framework for Malware Analytics

The malware and benign executables are reverse engineered applying context-sensitive data mining techniques and reverse engineering tools. This mining process is a critical step and includes rule generation and knowledge discovery validation. The extracted features obtained through the above process are constantly monitored and validated using deep monitoring and cross-system validation process. This is done to overcome the challenges posed by the adversary. Big data analysis tools and techniques make feature extraction and machine learning implementation more efficient and effective. Here, the heterogeneous data is handled using data categorization techniques. Also, we apply feature-based cleansing to make categorical homogeneous data. The training and testing dataset is passed to the machine learning algorithms or classifiers. Here, we apply adversarial defense and algorithmic biasness defense to mitigate the effects on the decision-making process. The final classification result is passed to the detection and alert system, which further handles the necessary steps to keep the system protected against any adversary attacks.

8.9 Summary

A survey of the most relevant aspects for malware analytics, mainly data mining, machine learning, and big data, has been reported in this work. It is showed how malware analysis had been done using prevalent techniques in data mining and machine learning. Big data comes into play between these two fields to allow the intelligent processing of massive data sets. Big data being a relative term, the available dataset would be considered big enough if the current typical systems cannot process it to analyze further to get some insights. Having Big data frameworks (e.g., Hadoop and Spark) in hand is highly recommended to fuse into the data mining and machine learning field to better program efficiency and remove the performance bottleneck. Malware analytics that includes handling executable, raw data, or traffic data to classification and detection may not have an excellent detection rate. Still, the performance would not be degraded below an acceptable bench-marking efficiency rate if we wisely use the fusion of the three rapid booming technologies: data mining, machine learning, and big data. The proposed framework is a step towards addressing open issues and challenges.

The scope of this survey is limited to analyzing the current status of the malware research highlighting the issues, and proposing a possible solution. However, it does not discuss issues like verifying the maliciousness of an input file, dealing with label uncertainty, and adaptive learning, which will be addressed in future editions. Some of the other future works to mention would be continuous research to capture code obfuscation techniques done by bad guys or malware writers to prevent malware detection. Another would be the constant improvement of current malware analysis and detection approaches. It is always a bonus gain of performance if we use distributed feature selection and machine learning model implementation. Future works should be using both, not the only one.

9 Conclusion and Future Work

Ransomware is continuously disrupting individuals to corporate networks demanding a huge ransom to give back the original unencrypted files. Social engineering techniques have been the major way to get into the victim's machine. Malware writers exploit user's urgency, need, and sentiments to trick them into clicking malicious links or attachments from where they start deploying the malicious software or payloads. Apart from this, malware writers exploit various vulnerabilities in software tools, network/system protocols, and APIs. Many known vulnerabilities have a patch update available, but users often ignore them and become prey to malware attacks. In contrast, the zero-day vulnerabilities allow the bad guys to craft malicious exploits and launch sophisticated attacks which even a security system or network may not detect for days to even months. Chainalysis [38] reported the total amount paid by ransomware victims increased by 336% in 2020. The attackers were able to gain nearly $370 million worth of cryptocurrency. According to the IBM Threat Intelligence Index report, 23% of incidents are ransomware compared to 2019 [24]. The most recent ransomware attack at Colonial Pipeline network forced the company to close down operations and freeze IT systems proactively. It is reported that they had to pay a huge ransom of nearly five million to get the network back and running [15]. Recent ransomware attacks are motivated to gain profit and cause damage sponsored by various underground state actors. In this study, I mainly focused on advanced reverse engineering with static and dynamic analysis of various ransomware families and various machine learning techniques.

This dissertation draws the following conclusions: I proposed an AI-powered ransomware detection framework using the techniques of reverse engineering, hybrid analysis, and machine learning. I started with two-level static analysis using DLL and assembly level code segments and proposed an initial framework for ransomware detection. Though the results were promising, I did an empirical study with three levels adding function call code segments to the initial two mentioned earlier along with NLP techniques. The developed models had sound performance but could not catch the dynamic behavior of malware executables. I proposed a more robust ransomware detection framework using hybrid analysis, behavior profiling, advanced reverse engineering, and AI techniques from continuous study and analysis. Leveraging dynamic binary instrumentation tool PIN, Cuckoo sandbox environment, and Ghidra framework, I generated a distinguishing feature dataset and achieved high accuracy and low false-positive rate. Association rule mining and Ghidra's disassembly contributed to the existing analysis by other approaches to build unique behavioral multi-level chains specific to ransomware. Collectively this contributes to creating unique Yara rules which researchers and the security community can use.

I performed a deep forensic analysis of crypto-ransomware using hybrid multi-level profiling. I adopted a unique approach of behavioral chaining along with association rule mining and AI techniques. Hybrid multi-level inspection at DLL, function call, and assembly revealed unique behavioral chains that help create unique ransomware signatures and a distinguishing dataset for the machine learning model. This approach is validated with experiments where I achieved high accuracy with low false positives. Results show that one machine learning algorithm achieved the highest accuracy of 99.72% and a false positive rate of 0.003 with two class datasets. Experiments done at multi-class malware families also revealed a reasonable accuracy rate (94.6%) with a very low false-positive rate of 0.001. Ransomware behavioral profiling chain ratio is a novel approach to identify ransomware binary, and it shows significant detection accuracy.

Malware analytics presents the current status of the malware research, challenges, and methods used to overcome those challenges using data mining, machine learning, and big data perspectives. I have considered these three perspectives because

of their extensive computation value, mostly fused to solve a wide range of problems from security to medical, finance, and industry. I also proposed a framework to overcome the challenges and open issues prevalent in malware analytics.
A periodic evaluation of chain components and automating this task will make the detection framework more robust in future work. This work can be upgraded in the future to deal with adversarial AI. Further analysis and experiments can be done using a wide range of ransomware families. To improve the performance of this framework, one possible direction for extending this work will be to use cloud computing with parallel processing capabilities. Thus, research pursued in this dissertation- a complex combination of AI-ML and hybrid forensics will provide an important and critical direction for future malware research and analysis.

Bibliography

[1] 2020 q1 threat report welivesecurity. https://www.welivesecurity.com/wp-content/uploads/2020/04/ESET_Threat_Report_Q12020.pdf. Accessed -on July 20, 2020.

[2] 2020 sonicwall cyber threat report. https://www.sonicwall.com/resources/2020-cyber-threat-report-pdf/. Accessed on July 20, 2020.

[3] 2021: Volumetric ddos attacks rising fast. https://blogs.akamai.com/2021/03/in-our-2020-ddos-retrospective. Accessed on May 20, 2021.

[4] 22 texas towns hit with ransomware attack in 'new front' of cyberassault. https://www.npr.org/2019/08/20/752695554/23-texas-towns-hit-with-ransomware-attack-in-new\protect\discretionary{\char\hyphenchar\font}{}{}front-of-cyberassault. Accessed: September-10, 2019.

[5] Apache spark mlib. https://spark.apache.org/mllib/.

[6] Apache spark rdd. https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm. 2019.

[7] Apt security: What are advanced persistent threats? https://securitytrails.com/blog/advanced-persistent-threats-apt. Accessed on May 30,2021.

[8] Bayesian network classifiers in weka. https://www.cs.waikato.ac.nz/~remco/weka.bn.pdf. May, 2008.

[9] Botnet taxonomy. https://sites.cs.ucsb.edu/~kemm/courses/cs595G/TM06.pdf. Accessed: October-30, 2019.

[10] Capstone the ultimate disassembler. https://www.capstone-engine.org/. Accessed on June 15, 2020.

[11] City of tulsa's online services disrupted in ransomware incident. https://www.bleepingcomputer.com/news/security/city-of-tulsas-online-services-disrupted-in-/ransomware-incident/. Accessed on May 21, 2021.

[12] Class adaboostm1. http://weka.sourceforge.net/doc.dev/weka/classifiers/meta/AdaBoostM1.html.

[13] Class logistic. http://weka.sourceforge.net/doc.dev/weka/cl-assifiers/functions/Logistic.html.

[14] Class smo. http://weka.sourceforge.net/doc.dev/weka/cl-assifiers/functions/SMO.html.

[15] Colonial pipeline attack: Everything you need to know. https://www.zdnet.com/article/colonial-pipeline-ransomware-attack\-everything-you-need-to-know/. Accessed on May 11, 2021.

[16] Command-and-control explained. https://www.paloaltonetworks.com/cyberpedia/command-and-control-explained. Accessed: October-28, 2019.

[17] Cuckoo automated malware analysis. https://cuckoosandbox.org. Accessed on May 22, 2020.

[18] Dc police confirms cyberattack after ransomware gang leaks data. https://www.bleepingcomputer.com/news/security/dc-police-confirms-cyberattack-after-rans\omware-gang-leaks-data/. Accessed on May 11, 2021.

[19] Flsalloc function. https://docs.microsoft.com/en-us/windows/win32/api/fibersapi/nf-fibersapi-flsalloc. Accessed on June 20,2020.

[20] Garmin begins recovery from ransomware attack. https://www.bbc.com/news/technology-53553576. Accessed on July 28, 2020.

[21] Ghidra. https://www.nsa.gov/resources/everyone/ghidra/. Accessed on June 22, 2020.

[22] How a drive-by download attack locked down entire city for 4 days. https://thehackernews.com/2017/10/drive-by-download-ransomware.html. Accessed: October-28, 2019.

[23] How to adapt to the new threat environment. https://home.kpmg/xx/en/home/insights/2020/05/rise-of-ransomware-during-covid-19.html. Accessed on July 10, 2020.

[24] Ibm x-force threat intelligence index. https://www.ibm.com/security/data-breach/threat-intelligence. Accessed on May 20,2021.

[25] Ida pro. https://hex-rays.com/. Accessed on June 15, 2020.

[26] M-trends 2020- fireeye. https://content.fireeye.com/m-trends/rpt-m-trends-2020. Accessed on July 20, 2020.

[27] Malwarebytes. https://www.malwarebytes.com/.

[28] Microsoft digital defense report 2020: Cyber threat sophistication on the rise. https://www.microsoft.com/security/blog/2020/09/29/microsoft-digital-defense-report-2020\-cyber-threat-sophistication-rise/. Accessed on Feb 22,2021.

[29] Microsoft windows smb server (ms17-010) vulnerability. https://www.cirt.gov.bd/microsoft-windows-smb-server-ms17-010-vulnerability/. Accessed on July 10,2020.

[30] Pe-parse tool. https://github.com/trailofbits/pe-parse. Accessed on May 05, 2020.

[31] pe-parser. https://github.com/trailofbits/pe-parse.

[32] Pin tool. https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html. Accessed on July 22, 2019.

[33] Ransomware. https://www.knowbe4.com/ransomware. Accessed on October 24, 2019.

[34] Ransomware 2018-2020 - kaspersky lab. https://media.kasperskycontenthub.com/wp-content/uploads/sites/100/2020/05/12075747/KSN-article_Ransomware-in-2018-2020-1.pdf. Accessed on July 20, 2020.

[35] Ransomware hits hundreds of dentist offices in the us. https://www.zdnet.com/article/ransomware-hits-hundreds-of-dentist-offices-in-the-us. Accessed: September-10, 2019.

[36] Ransomware: How to prevent being attacked and recover after an attack. https://www.backblaze.com/blog/complete-guide-ransomware/. Accessed: April-5, 2019.

[37] Ransomware is now the biggest cybersecurity threat. https://www.zdnet.com/article/ransomware-is-now-the-top-cybersecurity-threat-warns-kaspersky/. Accessed on July 20, 2020.

[38] Ransomware update: Newly uncovered addresses reveal $21m worth of new 2020 ransomware payments. https://blog.chainalysis.com/reports/ransomware-update-newly-uncovered-addresses-reveal\-21m-worth-of-new-2020-ransomware-payments. Accessed on May 20,2021.

[39] A record year for enterprise threats. https://documents.trendmicro.com/assets/rpt/rpt-2016-annual-security-roundup\-a-record-year-for-enterprise-threats.pdf. Accessed on May 20, 2018.

[40] Risksense spotlight report exposes top vulnerabilities used in enterprise ransomware attacks. https://risksense.com/press_release/risksense-spotlight-report-exposes-top-vulnerabilities\-used-in-enterprise-ransomware-attacks/. Accessed on July 10,2020.

[41] Sophos 2020 threat report. https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophoslabs-uncut-2020-threat-report.pdf. Accessed on July 20, 2019.

[42] Sophos 2020 threat report. https://www.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/sophos-the-state-of-ransomware-2020-wp.pdf. Accessed on July 20, 2019.

[43] thezoo - project created to make the possibility of malware analysis open. https://github.com/ytisf/theZoo.

[44] Thezoo, make the possibility of malware analysis open and available to the public. https://github.com/ytisf/theZoo.

[45] Virus total. https://www.virustotal.com/gui/home/upload. Accessed on June 22, 2020.

[46] Virustotal. https://www.virustotal.com/en/.

[47] Wastedlocker ransomware: Abusing ads and ntfs file attributes. https://labs.sentinelone.com/wastedlocker-ransomware-abusing-ads-and-ntfs-file-attributes/. Accessed on July 28, 2020.

[48] Weka 3: Data mining software in java. https://www.cs.waikato.ac.nz/ml/weka/. 2018.

[49] What is wannacry ransomware and why is it attacking global computers? https://www.theguardian.com/technology/2017/may/12/nhs-ransomware-cyber-attack-what-is-wanacrypt0r-20. Accessed on June 20,2020.

[50] What to do if you're infected by ransomware. https://www.tomsguide.com/us/ransomware-what-to-do-next,news-25107.html. Accessed: June-16, 2017.

[51] Yara rules. `http://virustotal.github.io/yara/`. Accessed on July 10, 2020.

[52] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499. Citeseer, 1994.

[53] Manaar Alam, Sayan Sinha, Sarani Bhattacharya, Swastika Dutta, Debdeep Mukhopadhyay, and Anupam Chattopadhyay. Rapper: Ransomware prevention via performance counters. *arXiv preprint arXiv:2004.01712*, 2020.

[54] Hyrum S Anderson, Anant Kharkar, Bobby Filar, and Phil Roth. Evading machine learning malware detection. *Black Hat*, 2017.

[55] Nicoló Andronio, Stefano Zanero, and Federico Maggi. Heldroid: Dissecting and detecting mobile ransomware. In *International Symposium on Recent Advances in Intrusion Detection*, pages 382–404. Springer, 2015.

[56] Abdullahi Arabo, Remi Dijoux, Timothee Poulain, and Gregoire Chevalier. Detecting ransomware using process behavior analysis. *Procedia Computer Science*, 168:289–296, 2020.

[57] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[58] Amin Azmoodeh, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE Transactions on Sustainable Computing*, 4(1):88–95, 2018.

[59] Alessandro Bacci, Alberto Bartoli, Fabio Martinelli, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio. Impact of code obfuscation on android malware detection based on static and dynamic analysis. In *ICISSP*, pages 379–385, 2018.

[60] Seong Il Bae, Gyu Bin Lee, and Eul Gyu Im. Ransomware detection using machine learning algorithms. *Concurrency and Computation: Practice and Experience*, page e5422, 2019.

[61] Manoj Basnet, Subash Poudyal, Mohd Ali, Dipankar Dasgupta, et al. Ransomware detection using deep learning in the scada system of electric vehicle charging station. *arXiv preprint arXiv:2104.07409*, 2021.

[62] L Breiman. Random forests machine learning. 45: 5–32. *View Article PubMed/NCBI Google Scholar*, 2001.

[63] Raymond Canzanese, Spiros Mancoridis, and Moshe Kam. System call-based detection of malicious processes. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 119–124. IEEE, 2015.

[64] CAP. Malware types. `https://cybersecurity1hub.com/malwares/`. accessed April 28, 2019.

[65] Carlos Cepeda, Dan Lo Chia Tien, and Pablo Ordóñez. Feature selection and improving classification performance for malware detection. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pages 560–566. IEEE, 2016.

[66] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[67] Mozammel Chowdhury, Azizur Rahman, and Rafiqul Islam. Malware analysis and detection using data mining and machine learning classification. In *International Conference on Applications and Techniques in Cyber Security and Intelligence*, pages 266–274. Springer, 2017.

[68] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. Shieldfs: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 336–347, 2016.

[69] Gregory F Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

[70] Zhihua Cui, Fei Xue, Xingjuan Cai, Yang Cao, Gai-ge Wang, and Jinjun Chen. Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics*, 14(7):3187–3196, 2018.

[71] Andrew Dinh, Daniel Brill, Yaohang Li, and Wu He. Malware sequence alignment. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pages 613–617. IEEE, 2016.

[72] Farnood Faghihi and Mohammad Zulkernine. Ransomcare: Data-centric detection and mitigation against smartphone crypto-ransomware. *Computer Networks*, 191:108011, 2021.

[73] Yujie Fan, Yanfang Ye, and Lifei Chen. Malicious sequential pattern mining for automatic malware detection. *Expert Systems with Applications*, 52:16–25, 2016.

[74] Justin Ferguson and Dan Kaminsky. *Reverse Engineering Code with IDA Pro*. Syngress, 2008.

[75] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156. Bari, Italy, 1996.

[76] Ibrahim Ghafir, Mohammad Hammoudeh, Vaclav Prenosil, Liangxiu Han, Robert Hegarty, Khaled Rabie, and Francisco J Aparicio-Navarro. Detection of advanced persistent threat using machine-learning correlation analysis. *Future Generation Computer Systems*, 89:349–359, 2018.

[77] Ibrahim Ghafir and Vaclav Prenosil. Dns traffic analysis for malicious domains detection. In *2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 613–918. IEEE, 2015.

[78] Nikolai Hampton, Zubair Baig, and Sherali Zeadally. Ransomware behavioural analysis on windows platforms. *Journal of information security and applications*, 40:44–51, 2018.

[79] William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. Dl4md: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Mining (DMIN)*, page 61. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016.

[80] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1507–1515. ACM, 2017.

[81] Jinsoo Hwang, Jeankyung Kim, Seunghwan Lee, and Kichang Kim. Two-stage ransomware detection using dynamic analysis and machine learning techniques. *Wireless Personal Communications*, 112(4):2597–2609, 2020.

[82] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[83] Firoz Khan, Cornelius Ncube, Lakshmana Kumar Ramasamy, Seifedine Kadry, and Yunyoung Nam. A digital dna sequencing engine for ransomware detection using machine learning. *IEEE Access*, 8:119710–119719, 2020.

[84] Amin Kharaz, Sajjad Arshad, Collin Mulliner, William Robertson, and Engin Kirda. {UNVEIL}: A large-scale, automated approach to detecting ransomware. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 757–772, 2016.

[85] Youngjoon Ki, Eunjin Kim, and Huy Kang Kim. A novel approach to detect malware based on api call sequence analysis. *International Journal of Distributed Sensor Networks*, 11(6):659101, 2015.

[86] Hyunji Kim, Jaehoon Park, Hyeokdong Kwon, Kyoungbae Jang, and Hwajeong Seo. Convolutional neural network-based cryptography ransomware detection for low-end embedded processors. *Mathematics*, 9(7):705, 2021.

[87] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. Paybreak: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 599–611, 2017.

[88] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 533–537. IEEE, 2018.

[89] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learning for classification of malware system call sequences. In *Australasian Joint Conference on Artificial Intelligence*, pages 137–149. Springer, 2016.

[90] Rafał Kozik, Marek Pawlicki, and Michał Choraś. Cost-sensitive distributed machine learning for netflow-based botnet activity detection. *Security and Communication Networks*, 2018, 2018.

[91] Jin Li, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-an, and Heng Ye. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225, 2018.

[92] Zitong Li, Xiang Cheng, Lixiao Sun, Ji Zhang, and Bing Chen. A hierarchical approach for advanced persistent threat detection with attention-based graph neural networks. *Security and Communication Networks*, 2021, 2021.

[93] Tariq Mahmood and Uzma Afzal. Security analytics: Big data analytics for cybersecurity: A review of trends, techniques and tools. In *2013 2nd national conference on Information assurance (ncia)*, pages 129–134. IEEE, 2013.

[94] Andrew Honig Michael Sikorski. Practical malware analysis. *No starch press*, (12), 2012.

[95] Tom M Mitchell. Machine learning (mcgraw-hill international editions computer science series). 1997.

[96] Daniel Morato, Eduardo Berrueta, Eduardo Magaña, and Mikel Izal. Ransomware early detection by the analysis of file sharing traffic. *Journal of Network and Computer Applications*, 124:14–32, 2018.

[97] Ramkumar Paranthaman and Bhavani Thuraisingham. Malware collection and analysis. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 26–31. IEEE, 2017.

[98] Subash Poudyal, Zahid Akhtar, Dipankar Dasgupta, and Kishor Datta Gupta. Malware analytics: review of data mining, machine learning and big data perspectives. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 649–656. IEEE, 2019.

[99] Subash Poudyal and Dipankar Dasgupta. Ai-powered ransomware detection framework. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1154–1161. IEEE, 2020.

[100] Subash Poudyal, Dipankar Dasgupta, Zahid Akhtar, and Kishor Gupta. A multi-level ransomware detection framework using natural language processing and machine learning. In *14th International Conference on Malicious and Unwanted Software "MALCON 2019" (MALCON 2019)*, Nantucket, USA, 2019. IEEE.

[101] Subash Poudyal, Kishor Datta Gupta, and Sajib Sen. Pefile analysis: a static approach to ransomware analysis. *Int J Forens Comput Sci*, 1:34–39, 2019.

[102] Subash Poudyal, Kul Prasad Subedi, and Dipankar Dasgupta. A framework for analyzing ransomware using machine learning. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1692–1699. IEEE, 2018.

[103] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[104] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE, 2015.

[105] Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin RB Butler. Cryptolock (and drop it): stopping ransomware attacks on user data. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pages 303–312. IEEE, 2016.

[106] Venkatesh Gauri Shankar, Mahesh Jangid, Bali Devi, and Shikha Kabra. Mobile big data: malware and its analysis. In *Proceedings of First International Conference on Smart System, Innovations and Computing*, pages 831–842. Springer, 2018.

[107] Shaila Sharmeen, Yahye Abukar Ahmed, Shamsul Huda, Bari Ş Koçer, and Mohammad Mehedi Hassan. Avoiding future digital extortion through robust protection against ransomware threats using deep learning based adaptive approaches. *IEEE Access*, 8:24522–24534, 2020.

[108] Saiyed Kashif Shaukat and Vinay J Ribeiro. Ransomwall: A layered defense system against cryptographic ransomware attacks using machine learning. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, pages 356–363. IEEE, 2018.

[109] Rami Sihwail, Khairuddin Omar, and KA Zainol Ariffin. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *International Journal on Advanced Science, Engineering and Information Technology*, 8(4-2):1662, 2018.

[110] Michael Sikorski and Andrew Honig. *Practical malware analysis: the hands-on guide to dissecting malicious software.* no starch press, 2012.

[111] Sanggeun Song, Bongjoon Kim, and Sangjun Lee. The effective ransomware prevention technique using process monitoring on android platform. *Mobile Information Systems*, 2016, 2016.

[112] Milan Stevanovic. Linux toolbox. In *Advanced C and C++ Compiling*, pages 243–276. Springer, 2014.

[113] Yuki Takeuchi, Kazuya Sakai, and Satoshi Fukumoto. Detecting ransomware using support vector machines. In *Proceedings of the 47th International Conference on Parallel Processing Companion*, pages 1–6, 2018.

[114] R Vinayakumar and KP Soman. Deepmalnet: evaluating shallow and deep networks for static pe malware detection. *ICT express*, 4(4):255–258, 2018.

[115] Wei Wang, Mengxue Zhao, and Jigang Wang. Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. *Journal of Ambient Intelligence and Humanized Computing*, 10(8):3035–3043, 2019.

[116] Wen-Chieh Wu and Shih-Hao Hung. Droiddolphin: a dynamic android malware detection framework using big data and machine learning. In *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, pages 247–252. ACM, 2014.

[117] Fei Xiao, Zhaowen Lin, Yi Sun, and Yan Ma. Malware detection based on deep learning of behavior graphs. *Mathematical Problems in Engineering*, 2019, 2019.

[118] Yus Kamalrul Bin Mohamed Yunus and Syahrulanuar Bin Ngah. Review of hybrid analysis technique for malware detection. In *IOP Conference Series: Materials Science and Engineering*, volume 769, page 012075. IOP Publishing, 2020.

[119] Ding Yuxin and Zhu Siyi. Malware detection based on deep learning algorithm. *Neural Computing and Applications*, 31(2):461–472, 2019.